Converting Handwritten Mathematical Expressions into LaTeX

Amit Schechter amitsch
Norah Borus nborus
William Bakst wbakst

December 15, 2017

1 Introduction

Recently there has been an increase in the usage of digital documents in academia. Although there has been work done to convert physical documents into their digital counterparts, most of it is limited to natural language. Digitizing mathematical expressions is becoming more necessary such as when submitting a problem set or writing a research paper, but many recognition systems fail to accurately convert special mathematical symbols such as μ or ϕ since they are not generally in natural language documents. Additionally, natural text converters fail to handle complex mathematical structures and relationships (e.g. fractions, summations, subscripts, etc.).

For applications such as writing problem sets and papers, many people often first hand write a draft that they later type up manually. LaTeX has a high learning curve, and the process of typing up mathematical documents is time consuming. This paper will explore the different approaches that we took to build a system that classifies handwritten mathematical expressions into their digital counterparts, which can be used for future real-time conversion applications.

The input of our system is a trace of a handwritten expression. We use an SVM for segmenting traces into characters. We then use a NN for character recognition and heuristics to analyze the structure of the expression. The output is the LaTeX representation of the expression.

For CS221, we work on a related project [18]. The focus in CS229 is the use of a Support Vector Machine and its features for character grouping and Neural Networks and Random Forest for character recognition. For CS221 we focused on Beam Search and Neural Networks for character grouping and a Support Vector Machine with Histogram of Gradients for character recognition. Since structural analysis currently uses simple heuristics, we share it across both projects.

2 Related Work

In the field of text recognition, we see two common approaches: online and offline recognition [5]. In online recognition, the writing is usually done on a digital medium such as a designated application on a tablet. In offline recognition, the writing is often done on a non-digital medium such as a piece of paper [14]. Naturally, online recognition obtains more information, such as the order of writing the characters, and it therefore obtains higher accuracy compared to offline recognition [17].

Some previous work has been done on converting typed PDF documents (rather than handwritten ones) into their LaTeX representation [1]. Dividing the problem into segmentation, character recognition, and structural analysis often results in very high accuracy for this setting. Since typed PDF documents have far less variability in the data, this problem is also far less complex, and we will address the variability of the data in our implementation and analysis.

Detexify is an application that converts handwritten characters into their LaTeX counterpart. Detexify is quite accurate when classifying characters, and it displays a scrollable list of likely classifications, which is nice. However, Detexify is limited to single character recognition, making it difficult to find the LaTeX equivalent for more difficult expressions such as $\sum_{i=1}^{m}$.

Another online application that converts handwritten data into LaTeX is Visual Objects Web Equation, also called MyScript. This application allows users to write expressions on a pad in their browser while it converts that data into LaTeX in real time. MyScript works very well, perfectly classifying the majority of our simple and even most of our tougher examples, but still failing on some of our more difficult examples.

3 Datasets

We use the "Handwritten Mathematical Expressions" dataset from Kaggle [10]. It includes around 11,000 inkml files representing handwritten equations (72,373 single character images) mapped to the corresponding ground truth LaTeX expression.

Each inkml file includes traces of a single expression, ground truth segmentation into characters, and ground truth LaTeX (including mapping of single characters to their LaTeX representation). We split the data into 80% train, 10% dev, and 10% test. We convert every trace into its corresponding normalized gray-scale image. [1]

Example:

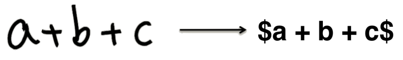

Figure 1: Handwritten Sample

The distribution of characters in the dataset is far from uniform (see figure 2). To provide more training examples for uncommon characters, we perform data augmentation using random elastic transformations [6], [7].

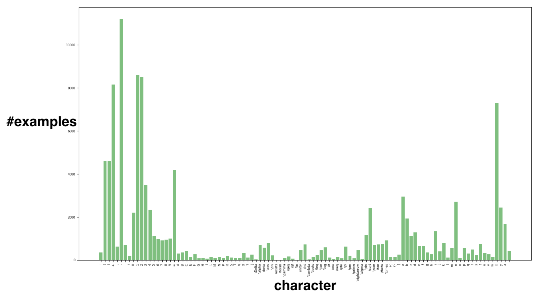

Figure 2: Distribution of characters in dataset

4 Methods

4.1 Neural Net

We implement an NN (see figure 3) with a single hidden layer (with 400 hidden units), using sigmoid activation for the hidden layer, and softmax activation for the output layer. The sigmoid function is defined as $g(z) = \frac{1}{1+e^{-z}}$. It transforms the feature vector (input array) into a real number $x \in [0,1]$.

The softmax hypothesis outputs the estimated probability that $p(y = i|x;\theta)$, where x is the output of the sigmoid function, and i that represents one of the 101 classes in the dataset ($i = 1, ..., 101$). Our prediction is the class with the maximum probability. We perform mini-batch gradient descent (with batch size 1000), using the average cross entropy loss over the training examples as our cost function:

$$J(W^{[1]}, W^{[2]}, b^{[1]}, b^{[2]}) = \frac{-1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} y_k^{(i)} log(\hat{y}_k^{(i)})$$

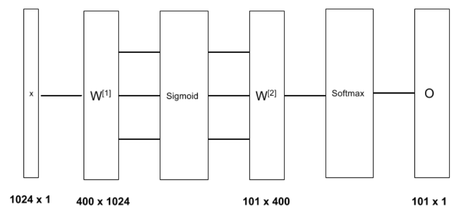

Figure 3: NN Architecture

4.2 Convolutional Neural Net

We use a CNN (see figure 4) of the form CONV - ReLU - POOL - CONV- ReLU - POOL - FC - ReLU - FC - ReLU - FC [11]. The two (feature-extracting) Convolutional layers use 5×5 kernels, the two pooling layers perform max pooling over a (2, 2) window, and the three fully-connected layers use an affine operation $y = Wx + b$. We use ReLU as the activation function for each layer (recall ReLU is defined as the positive part of its argument $f(x) = max(0,x)$). The output of the CNN is the score assigned to each LaTeX symbol. Our prediction is the symbol with the maximum score. We perform stochastic gradient descent, also using cross-entropy loss as our cost function.

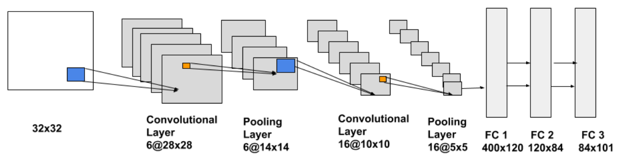

Figure 4: CNN Architecture

4.3 Random Forest

We decided to try the random forest classifier [9], which was not covered in class. It is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. It has proved to be very effective in several fields, including handwritten digit recognition.

4.4 Support Vector Machine (SVM)

Binary SVM is a linear classifier that attempts to find support vector machines that best separate the data (see dotted lines in figure 5). It does it by optimizing with respect to the hinge loss function:

[1] For the segmentation results presented in this paper, we train on a subset of the dataset (roughly 10% of the complete dataset), because we did not have enough computing resources to train on the complete dataset.

$L_{hinge} = \frac{1}{N}\sum_i [1 - z_i]_+ = max\{1 - z_i, 0\}$ for binary classification and
$L_{multi} = \frac{1}{N}\sum_i \sum_{j \neq y_i} [max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$ for multi class SVM. We use an SVM implementation from scikit [12].

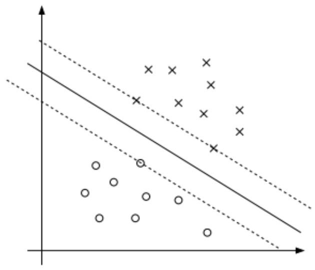

Figure 5: SVM geometric representation

4.5 Grouping (CGrp) Methods

In the grouping step, for each expression, the input is an array of strokes. The output is a list of lists of strokes grouped by character.

Our final implementation uses a binary SVM. We decided to use SVM since we were facing a classification task, where we had many examples of both classes (traces that go together and ones that do not). Following the CS229 notes, we hypothesized that an SVM will learn to differentiate between the classes and outperform a binary logistic classifier.

Based on our data analysis, we found that features describing the relationship between two strokes are the main indicators for grouping traces. For each pair of strokes, we extract features such as the inverse of the horizontal and vertical distance between the strokes. We also use an indicator whether or not one stroke is horizontally contained within the bounds of the other, and the inverse of distance between center of mass of the strokes. We use the inverse because we found that smaller distances indicate that strokes are likely to be grouped. We previously experimented with additional features such as angle between center of mass of traces, which did not improve segmentation performance.

In addition to the SVM implementation, we experimented with a NN combined with beam search (which we focused on for our CS221 project). More details can be found in our CS221 final project paper [16].

4.6 Character Recognition (OCR) Methods

For the neural net, the input data is an array of length 1024 (the flattened version of the 32×32 normalized gray-scale character's pixel array) and the ground truth label is a one-hot representation of the character's true class. The output is the estimated probabilities of each LaTeX symbol being the character's correct category.

For the CNN, the input is a 32×32 normalized gray-scale pixel array of the character, and the output is score assigned to each LaTeX symbol class.

We decided to try the simple NN because we were inspired by the high performance of the NN implemented in Problem Set 4 on the MNIST dataset. We were inspired by the CS229 lecture and CS231N lecture notes to use a CNN, based on the understanding that CNNs often performs well on OCR.

An alternative approach we experimented with is the Random Forest Classifier [9]. We tuned the RF main parameters (the number of trees in the forest and the number of random features pre-selected in the splitting process) in order to analyze the correlation between the RF's performance and the relevant parameters.

4.7 Structural Analysis (SA) Methods

For structural analysis, the input is the trace data for each character and the corresponding LaTeX representation. These come from the CGrp and OCR steps. The output is the list of relationships between characters and the corresponding rebuilt LaTeX expression.

Our current implementation for structural analysis uses simple heuristics to determine the relationship between two characters. We found that this approach works quite well for clean data, since the relationships between characters is rather clear, but that these simple heuristics tend to fail on noisier data.

From our analysis, we found that basic features describing the location relationships between two characters works well in finding the relationships between characters. For each expression, we determine whether two characters have one of the relationships found in Figure 6. Specifically, we extract the minimum x and y coordinates as well as the maximum x and y coordinates to find a bounding box for each character. We then check for certain relationships such as complete containment by performing simple arithmetic on these bounding boxes. If we find that there is a certain relationship, we record it and continue. We also found that certain heuristics such as subscripts and superscripts were difficult to classify correctly since many handwritten samples had super and subscripts that were well outside of our predefined heuristic bounds. We are also still working on classifying fractions, which becomes more difficult as there are more terms in the numerator and denominator.

We also looked into using an SVM or an NN to help classify these relationships because we thought that there could be hidden features that would better classify these relationships. However, we were unable to find a good LaTeX parser. In order to train an SVM or NN, we would need to first parse the ground truth expressions into the ground truth relationships to create the correct training labels. Because writing a parser is tedious and not particularly related to Machine Learning, we decided to wait to implement this parser ourselves in the future. See future work for more.

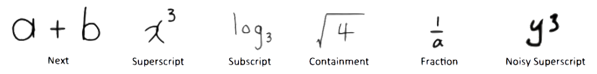

Figure 6: Different Relationships

5 Experiments/Results/Discussion

5.1 Grouping (CGrp) Results

To evaluate grouping, our primary metric is accuracy, measuring the number of correctly segmented characters out of the total number of characters (if multiple strokes of the same character were correctly grouped together but were also grouped with additional strokes, the character is considered mis-grouped).

CGrp using an SVM achieves 96% train accuracy and 65% test accuracy. From error analysis we noticed that the majority of errors come from strokes that should be grouped together but the SVM did not group. The SVM over-fits the training data, and we get similar results using different regularization constants. Due to the high variation in people's handwriting (see figure 7), we believe that to address this, we would have to train on many more examples, which requires additional computing resources or advanced optimization methods.

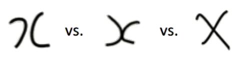

Figure 7: Variability in handwriting for the letter X

The binary NN used to classify images as valid versus invalid characters (which we focus on for CS221) achieves 97% train accuracy and 78% test accuracy (see figure 8), using learning rate 5 and regularization factor $\lambda = 0.0003$. These parameters were chosen by experimenting with multiple values.

True / Predicts	Positive	Negative
Positive	193	23
Negative	206	642

Figure 8: 2 classes NN confusion matrix

Using beam search and the binary NN to generate and select groupings, we achieve 64% accuracy. In contrast to the SVM implementation, in this setting, most errors come from characters that should not be grouped but are grouped together by the algorithm.

5.2 Character Recognition Results

Our evaluation metric is measuring the number of correctly classified images out of the total number of images fed to the CNN, NN and RF classifier (accuracy).

5.2.1 CNN and NN Results

The CNN achieves 86% train accuracy and 86% test accuracy. The NN achieves 91% train accuracy and 83% test accuracy. While the CNN does not overfit the data, the NN does. This is consistent with what we read in the CS231N lecture notes (it was noted that the full connectivity of regular neural nets often leads to overfitting).

Interestingly, the test accuracy was the same for both networks, even though the NN had a single hidden layer. We hypothesize that it might be because the neural net uses both softmax and sigmoid as the activation functions. ReLU units speed up training (as compared to softmax and sigmoid neurons that involve expensive operations such as exponentials), but ReLU units can be fragile during training [16].

We noticed in our error analysis that characters that were very similar in appearance (e.g. $\cdot$ [represented as \cdot] and . [dot]) were often mistaken for each other in the predictions: since the models take in isolated characters, our approach was not able to recognize the relationship between the character and it's neighbors in order to remedy such misclassifications. An end-to-end approach would likely not suffer from a similar problem (see future work).

5.2.2 Random Forest Results

We first tested the influence of max_features (the number random features pre-selected in the splitting process). We fixed the value of max_features and tested the recognition rate on a range of values for num_features. From our tests, setting max_features in the range [15,35] produces the highest dev accuracy. We then tested the influence of num_estimators (the number of trees in the forest). We fixed the value of num_estimators and tested the recognition rate on a range of values for max_features. From our tests, setting num_estimators in the range [250,350] the highest dev accuracy (see figure 9). Combined, setting max_features as 30 and num_estimators as 300 produced the highest test accuracy : 75%.

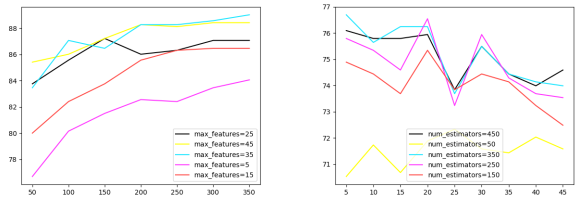

Figure 9: Recognition rates w.r.t num_estimators (left) and w.r.t to max_features(right)

5.2.3 Data Augmentation

We noticed that the characters that had the most test error belonged to classes that did not appear as commonly in the dataset (e.g. γ). From our error analysis, we decided to augment the available data for uncommon symbols using randomized elastic transformation in order to improve the recognition rate for these symbols (see figure 10).

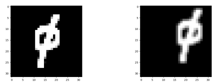

Figure 10: Elastic transformation of ϕ (sigma=60, alpha=500)

We tested the image recognition rate for the rare symbols specifically after data augmentation, and noted a marked improvement in the recognition rate (see figure 10).

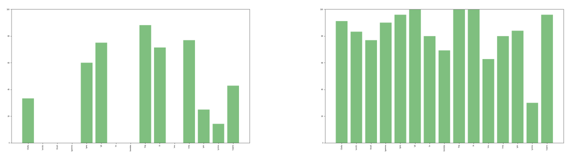

Figure 11: Dev accuracy of uncommon classes before (left) and after (right) augmentation

5.3 Structural Analysis (SA) Results

We evaluate structural analysis based on the total number of relationships in an expression that it correctly classifies. Currently, we are calculating the accuracy by hand, first determining the relationships in the correct label, then comparing it to the relationships defined by the SA module. Once we build a LaTeX parser (see Future Work), we will be able to test accuracy automatically.

Using our basic heuristics described earlier, we achieve 65% test accuracy (no training involved). The majority of our errors come from fractions with many terms, recursive containment and super/subscripting (Figure 6), as well as noisy and messy handwriting. In order to improve our accuracy, we believe that a Neural Net would work well since it could find hidden relationships between handwritten characters that are too difficult to manually define (see future works for more).

6 Conclusion/Future Work

6.1 Character Grouping Conclusion

CGrp achieved similar accuracy with SVM and and NN (beam search). The SVM implementation overfits, an issue which we believe we can solve by training on more data (computing resources is currently the limiting factor). Additionally, we've noticed that the errors for the SVM implementation and the NN one come from different sources, where SVM tends to keep strokes separate and NN tends to group them. Thus, we hope to further improve accuracy by weighting the outputs of both models to determine grouping.

6.2 Character Recognition Conclusion

We saw relatively high recognition rates across all algorithms used, with the CNN achieving the highest test accuracy. The single-layer NN overfits the data due to it's full-connectivity, but performs only slightly worse than the 5-layer CNN, and we hypothesize that this is because we use more stable/sophisticated activation functions in the neural net.

Because some parameter values are commonly used without empirical justifications (for example the number of trees in the forest is commonly arbitrarily fixed to 100), and because the RF ran to completion faster than the NN and CNN, we focused on the parameter tuning process to present a pragmatic approach on using the Random Forest for classification [15]. We believe that some of the errors come from high variability and illegible handwriting, which humans often fail to understand.

6.3 Structural Analysis Conclusion

For the scope of this project, we decided that writing a LaTeX parser was too tedious and unrelated to Machine Learning, so we focused primarily on the ML aspects of CGrp and OCR. In the future, we plan to write our own parser that will build a relationships map for each ground truth LaTeX expression. Using this map, we could accurately train an SVM and an NN using these labels and the character pairings as training data. We believe that these models will be able to find relationships that are too difficult to manually define as heuristics, performing with higher accuracy in turn.

6.4 End To End

Additionally, an end-to-end approach might be successful. Since grouping, classification, and structural analysis are closely related, we believe that a multiple layer CNN might be able to perform well on converting traces of whole expressions into their LaTeX representation (similar to Tesseract, Google's OCR engine).

7 Contributions

William (wbakst): Structural Analysis, SVM features for Character Groupings, paper writeup.
Norah (nborus): Character Recognition, paper writeup, data augmentation.
Amit (amitsch): Character Grouping, paper writeup.

8 References/Bibliography

[1] Chang, Joseph, Shrey Gupta, and Andrew Zhang. "Painfree LaTeX with Optical Character Recognition and Machine Learning." (2016).

[2] Thoma, Martin. "On-line Recognition of Handwritten Mathematical Symbols." arXiv preprint arXiv:1511.09030 (2015).

[3] Xuejin, Zhao, et al. "On-line recognition handwritten mathematical symbols." Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on. Vol. 2. IEEE, 1997.

[4] Bluche, Théodore. Mathematical Formula Recognition using Machine Learning Techniques. Diss. University of Oxford, 2010. APA

[5] Lu, Catherine, and Karanveer Mohan. "Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks." APA

[6] Rahaman, Nasim. "elastic_transform.py".

[7] Wong, Sebastien C., et al. "Understanding data augmentation for classification: when to warp?." Digital Image Computing: Techniques and Applications (DICTA), 2016 International Conference on. IEEE, 2016.

[8] Scikit. "Histogram of Oriented Gradients".

[9] Scikit. "Random Forest Classifier".

[10] Kaggle, Tatman, Rachael. "Handwritten Mathematical Expressions".

[11] Pytorch. "Neural Networks".

[12] Scikit. "Support Vector Machines".

[13] Trier, Øivind Due, Anil K. Jain, and Torfinn Taxt. "Feature extraction methods for character recognition-a survey." Pattern recognition 29.4 (1996): 641-662. APA

[14] Plamondon, Réjean, and Sargur N. Srihari. "Online and off-line handwriting recognition: a comprehensive survey." IEEE Transactions on pattern analysis and machine intelligence 22.1 (2000): 63-84.

[15] Simon Bernard, Laurent Heutte, Sébastien Adam. Using Random Forests for Handwritten Digit Recognition. Proceedings of the 9th IAPR/IEEE International Conference on Document Analysis and Recognition ICDAR'07, Sep 2007, Curitiba, Brazil. pp.1043-1047, 2007.

[16] CS231n Convolutional Neural Networks for Visual Recognition, cs231n.github.io/neural-networks-1/.

[17] LeCun, Yann. "The MNIST database of handwritten digits." http://yann. lecun. com/exdb/mnist/ (1998).

[18] Borus, Norah, Schechter Amit, Bakst William. "Converting Handwritten Mathematical Expressions into LaTeX", CS221. (2017).

[19] MyScript. //webdemo.myscript.com/views/math.html.