Handwritten digits recognition

Kristina Pomorišac
Department of Computing and Control Engineering
Faculty of Technical Sciences
Novi Sad, Serbia
kristipomo@gmail.com

***Abstract*—Converting handwritten formulas to LaTex is a challenging machine learning problem. An essential step in the recognition of mathematical formulas is the symbol recognition. In this paper you can learn how image manipulation, analysis, as well as optical character recognition via machine learning are effective features for recognizing mathematical symbols. Using OpenCV the program will first process and parse the image into the individual characters that will be recognized later. This program will also use OpenCV to understand the formatting of the handwritten expression, such as subscripts or superscripts for better parsing. The classifier is trained by using images extracted from XY coordinates of online data from the mnist database, which contains 60000 character samples. The final results are displayed on screen using matplotlib.**

Keywords—machine learning; symbol recognition; handwritten expression; mnist database;

I. Introduction

Most previous research on mathematical formula recognition has focused on online data, i.e. the writer’s strokes are recorded on a computer or tablet as a sequence of xy coordinates. Using such datasets presents many advantages when trying to recognize full formulas. Most importantly, strokes can be individually labelled using a GUI based data collection engine. This makes the location of each symbol easy to parse for a learning algorithm. The mnist database (downloadable at: http://yann.lecun.com/exdb/mnist/), which has become a standard for handwritten digits recognition..

On the other hand, the restriction of mathematical formula recognition to online data limits the usefulness of a mathematical formula recognition system. Transcribing LaTex expressions from handwritten sources can be a painful experience even for seasoned veterans. It would be very desirable for university students to be able to take pictures of their homework using their phone and have an engine return a typeset document. Potentially, such an engine could also automate mathematical note taking in classrooms so that students can focus on the content without the distraction of copying formulas.

In this paper you can discover the first steps towards fulfilling such an objective. Here you can see how OCR recognition is a highly effective approach to recognizing mathematical formulas and symbols extracted from datasets. The process is followed by present encouraging preliminary results of an OpenCV based pipeline that uses images taken from a phone.

II. Data preparation

The database used in the project is the mnist database (downloadable at: http://yann.lecun.com/exdb/mnist/), which has become a standard for handwritten digits recognition.This database has a training set of 60,000 examples, and a test set of 10,000 examples. It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. The data is stored in a very simple file format designed for storing vectors and multidimensional matrices. The digit images in the MNIST set were originally selected and experimented with by Chris Burges and Corinna Cortes using bounding-box normalization and centering. [1]

In order to create images from these XY coordinates, intermediate XY coordinates were first linearly extrapolated from consecutive XY pairs. Using the gathered XY data you can create the training data, validation data and test data sets. A properly scaled bitmap was then created from these XY coordinates. Afterwards the newly created bitmap is vectorized. The reason behind this is to sharpen the edges of the bitmap and make the symbols easier to be recognized. The gained difference is is visually described on Fig. 1.

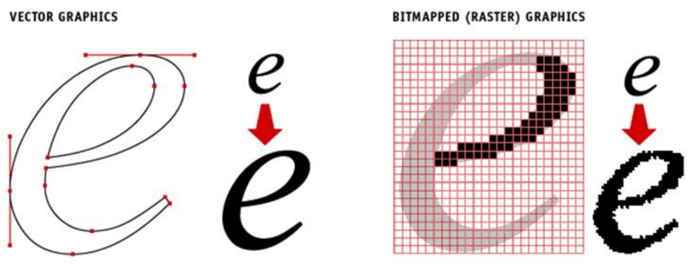


Fig. 1. Vector and bitmap example

Sigmoid function [2] is a mathematical function, having a characteristic “S” shaped curve or sigmoid curve. The use of this function is to calculate results gained after vectorizing the bitmap and to calculate the sigmoids prime. Sigmoid function is the algebraic form of sigmoid neuron. Later in this paper we will use the sigmoid neuron in neural networks for calculating the weights of the inputs.

III. Neural network

Neural networks and deep learning currently provide the best solutions to many problems in image recognition, handwritten digits recognition, speech recognition, and natural language processing.

Training examples from data preparation are used for the neural network to be able to recognize a large number of handwritten digits and then develop a system which can learn from those training examples. One of those examples is shown in Fig. 2. In other words, the neural network uses the examples to automatically infer rules for recognizing handwritten digits. Furthermore, by increasing the number of training examples, the network can learn more about handwriting, and so improve its accuracy.

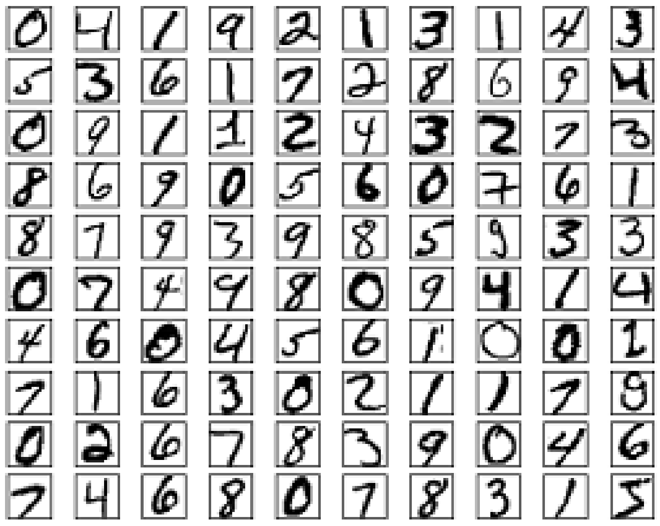


Fig. 2. Training example from mnist dataset

In this projects a class called "Network" is used to represents a neural network. The biases and weights in the Network object are all initialized randomly, using the Numpy np.random.randn function to generate Gaussian distributions with mean 0 and standard deviation 1. This random initialization gives our stochastic gradient descent algorithm a place to start from.

"CrossEntropyCost" is a function used in the "Network" for calculating costs of outputs. It is a measure of how well an output activation, matches the desired output

The core functions used for recognizing handwritten digits, as well as improving the machine learning process in the Network class are: stochastic gradient descent and backpropagation algorithm.

A. *Stochastic gradient descent*

One of the standard learning algorithm for neural networks is known as stochastic gradient descent. In practice, stochastic gradient descent is a commonly used and powerful technique for learning in neural networks and to speed up the learning process. Stochastic gradient descent works by randomly picking out a small number of randomly chosen training inputs.

In the project the stochastic gradient descent function is mainly described with parameters: training_data, variables epochs and evaulationData. The training data is a list of tuples (x, y) representing the training inputs and corresponding desired outputs. The variables epochs represent the number of epochs you use to train the inputs. If the optional argument evaulationData is supplied, then the program will evaluate the network after each epoch of training, and print out partial progress.

The code works as follows. In each epoch, it starts by randomly shuffling the training data, and then partitions it into sections of the appropriate size. This is an easy way of sampling randomly from the training data. Then for each section we apply a single step of gradient descent. This updates the network weights and biases according to a single iteration of gradient descent, using just the training data in the section. This is done with the function called "updateSection". This invokes the backpropagation algorithm, which is a fast way of computing the gradient of the cost function.

B. *Backpropagation algorithm*

The backpropagation algorithm [3] was originally introduced in the 1970s, but its importance wasn't fully appreciated until a famous 1986 paper by David Rumelhart, Geoffrey Hinton, and Ronald Williams. That paper describes several neural networks where backpropagation works far faster than earlier approaches to learning, making it possible to use neural networks to solve problems which had previously been insoluble. Today, the backpropagation algorithm is the workhorse of learning in neural networks.

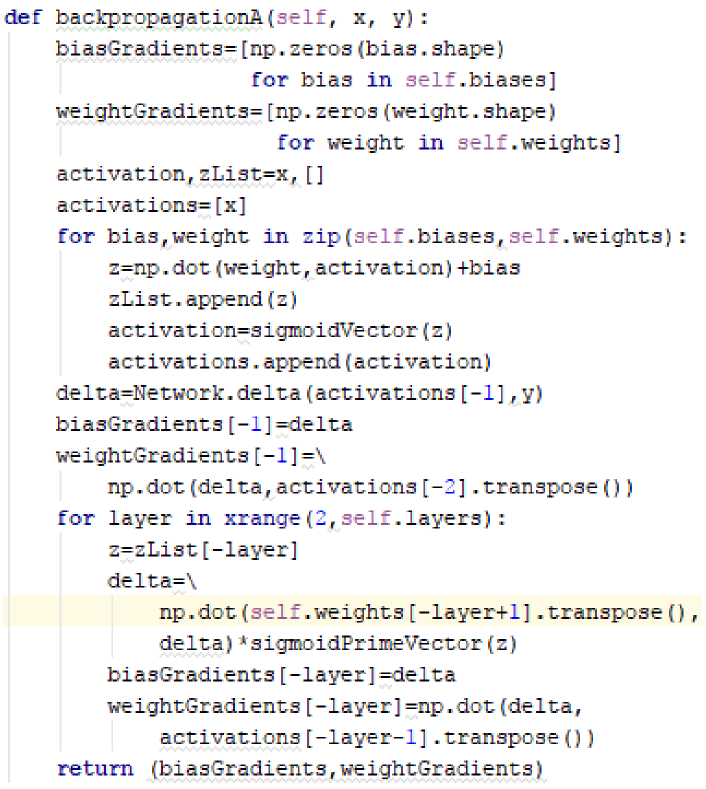
At the heart of backpropagation is an expression for the partial derivative $\partial C/\partial w$ of the cost function C with respect to any weight w (or bias b) in the network. The expression tells us how quickly the cost changes when we change the weights and biases. This algorithm gives us detailed insights into how changing the weights and biases changes the overall behaviour of the network.

In this project a full matrix based approach to backpropagation is used to implement stochastic gradient descent loops, Fig. 3. The idea is that instead of beginning with a single input vector, x, we can begin with a matrix X=[x1x2…xm] whose columns are the vectors. We forward-propagate by multiplying by the weight matrices, adding a suitable matrix for the bias terms, and applying the sigmoid function everywhere and backpropagate along similar lines.

IV. Image processing

Often the development of a computer vision project involves tweaking parameters of a technique to achieve the desired outcome. These parameters could be the thresholds of an edge detection algorithm or the brightness of an image, for instance. The computer vision project that is used is called "OpenCV" [4] .

OpenCV is the leading open source library for computer vision, image processing and machine learning, and now features GPU acceleration for real-time operation. OpenCV is released under a BSD license and hence it’s free for both academic and commercial use.



```
def backpropagationA(self, x, y):
    biasGradients=[np.zeros(bias.shape)
                   for bias in self.biases]
    weightGradients=[np.zeros(weight.shape)
                     for weight in self.weights]
    activation,zList=x,[]
    activations=[x]
    for bias,weight in zip(self.biases,self.weights):
        z=np.dot(weight,activation)+bias
        zList.append(z)
        activation=sigmoidVector(z)
        activations.append(activation)
    delta=Network.delta(activations[-1],y)
    biasGradients[-1]=delta
    weightGradients[-1]=\
        np.dot(delta,activations[-2].transpose())
    for layer in xrange(2,self.layers):
        z=zList[-layer]
        delta=\
            np.dot(self.weights[-layer+1].transpose(),
            delta)*sigmoidPrimeVector(z)
        biasGradients[-layer]=delta
        weightGradients[-layer]=np.dot(delta,
            activations[-layer-1].transpose())
    return (biasGradients,weightGradients)
```

Fig. 3. Backpropagation algorithm

On the images the following steps are applied by using OpenCV and open source libaries to segment the images into constituent symbols:

- Image blur to remove noise (OpenCV2 function: GaussianBlur)
- Invert every bit of an array (OpenCV2 function: bitwise_not)
- Adaptive threshold to binarize the image (OpenCV2 function: adaptiveThreshold)
- Creating Bounding boxes and circles for contours (OpenCV2 function: boundingRect)
- Segment image by finding connected components (OpenCV2 function: findContours)

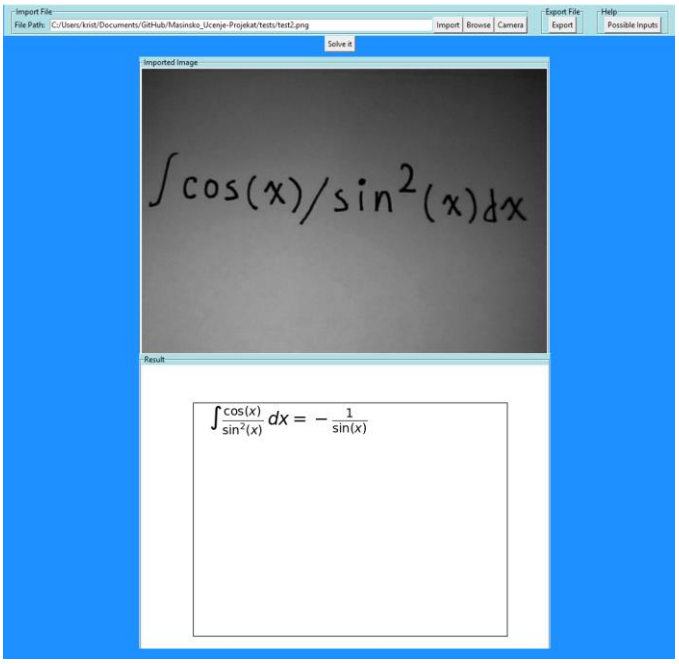
At first bounding lines are created around recognized characters. Afterwards we separate the characters into images with each character representing one image, or bounding box. A deep copy [5] constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original image. Once each symbol is separated in bounding boxes, GaussianBlur and findContours functions are applied.

Using OpenCV the program will first process and parse the image into the individual characters loaded from the mnist database that will be recognized later. The next step involves the use of a neural network to perform the Optical character recognition (OCR) operations, this allows us to recognize math symbols such as: integral, pi, sum, sqrt, si, and others. After the characters are parsed and formatted they will be reconstructed into computer legible code and fed into the sympy [6] or scipy [7] module in order to solve them, and create this projects calculator. Afterwards the results will be displayed on screen using matplotlib [8].

V. Calculator

The calculator created with this project is able to recognize handwritten mathematical symbols from the images that are imported from the system files or taken by the camera. Calculator provides simple and advanced mathematical functions. It can perform basic calculations such as addition, subtraction, multiplication, and division. And it can do scientific operations such as trigonometric, inverse trigonometric, calculus, and exponential functions.

The calculator receives parsed and formatted characters after the image has been processed. Once an image with mathematical symbol exists on the imported image side of the application and the user presses the button “Solve it”, the results will be displayed on screen using matplotlib as shown in Fig. 4. Appropriate error if the mathematical expression has no valid result. The recognized symbols are always printed on command prompt. Your results are always saved in the temp folder and can be exported for the future use.



File Path: C:/Users/krist/Documents/GitHub/Masinsko_Ucenje-Projekat/tests/test2.png

$$\int \frac{\cos(x)}{\sin^2(x)}\,dx = -\frac{1}{\sin(x)}$$

Fig. 4. Calculator

The application is not optimal, but to get your desired result the user can improve the application symbol recognition and detection by using a black while paper and dark, preferably black pan. A clean writing style is helpful as well.

Every new image made by using the application is saved and generates the training data from the scanned image of specific formatting. The network learn more about handwriting as we increase the number of training examples, this helps to improve the overall accuracy in detecting and recognizing the gives symbols

VI. Conclusion

It is an undeniable fact that it is easier to handwrite mathematical expressions and symbols than it is to type them into a computer. Despite the innumerable workarounds that have sprung up to ease the pain of inserting special mathematical symbols into digital documents, digital input of mathematical and scientific symbols remain rather tedious and difficult. Yet it is also undeniable that computers remain a far better solver of mathematical and scientific equations than humans ever will be, and thus in one way or another mathematical expressions must be input into digital form.

The application that was described in this paper is not ideal, but with the help of the mnist database and every new image added to it, the application learns and improves. With this the accuracy is increased. Backpropagation algorithm is a good option when you deal with huge amounts of data and you want to solve a supervised learning task. The reason for this is that for a complex neural network, the number of free parameters is very high.

Since this application has primarily two components, in the continuation two kinds of products will be independently investigated. The first, OCR as applied to mathematical formulas and the second, the processing of more intuitive forms into computer legible inputs and the solving of them.

The most complete software for mathematical formulas recognition is the “InftyProject” [9], a research project that produces software that can analyze scanned scientific and mathematic documents and output LATEX format mathematical formulas. Its features include the preservation of original layout via LATEX formatting as well as fairly good accuracy on less complicated expressions. This software and the application introduced in this paper share the same core feature, the OCR of mathematical symbols into computer legible form. The largest strength of this software is the ability to preserve the layout of the original document, enabling scanning of large documents with multiline components. Despite being the most complete of the software packages, “InftyProject” is still quite incomplete, in particular it struggles with more complex expressions, often missing variables and misinterpreting when the expressions grow long. Its largest drawback by far however, is its inability to process handwritten formulas, “InftyProject” is only capable of recognizing scanned documents, not handwritten symbols.

The next software, “JMathNotes” [10], is a proof of concept software that attempts to recognize digital handwritten mathematical formulas. Although it is incomplete the software nonetheless manages to impress as it recognizes distorted handwritten expressions. This software, however, along with most other softwares that recognize handwritten symbols, only allow for digital input and is incapable of reading physical handwritten inputs.

The second set of software, the ones that process and solve mathematical inputs, is far more established and is represented by its most famous example, “WolframAlpha” [11]. “WolframAlpha” is capable of taking in a staggering variety of digital mathematical inputs as well as natural language writing and convert them into computer legible form before solving and displaying the results. The features of “WolframAlpha” are too many to enumerate but the one feature that could be implement in this application are some basic commands that can be handwritten on paper that can be parsed by the program. For example if the command “plot” is beside an equation then the program would plot the equation, and if the command is “solve”, then the program would solve and display the equation.

References

[1] The mnist database, http://yann.lecun.com/exdb/mnist/

[2] Sigmoid function, http://mathworld.wolfram.com/SigmoidFunction.html

[3] Michael Nielsen, “Neural Networks and Deep Learning”, http://neuralnetworksanddeeplearning.com, May 2017

[4] OpenCV, http://opencv.org/

[5] Shallow and deep copy operations, https://docs.python.org/2/library/copy.html

[6] SymPy, http://www.sympy.org/en/index.html

[7] SciPy, https://www.scipy.org/

[8] Matplotlib, https://matplotlib.org/

[9] Infty Project, http://www.inftyproject.org/en/index.html

[10] JmathNotes, http://userpage.fu-berlin.de/~tapia/?page_id=269

[11] WolframAlpha, https://www.wolframalpha.com/