Recognition of Handwritten Mathematical Symbols

Bachelor's Project report for semester 8
(Mentor: Dr. Gaurav Harit)

Ajay Charan
UG201211002
Dept. of Electrical Engineering
IIT Jodhpur

Table of Contents

Abstract

Automatic recognition of mathematical expressions is one of the key vehicles in the drive towards transcribing documents in scientific and engineering disciplines into electronic form. This problem typically consists of two major stages, namely, symbol recognition and structural analysis. In this work, we will review most of the existing work with respect to the recognition process and also propose and analyze our methods for recognition. In particular, we try to put emphasis on the similarities and differences between systems. Moreover, some important issues in mathematical expression recognition will be addressed in depth. All these together serve to provide a clear overall picture of this research area.

Motivation and Scope

With the very rapid increase of Internet based learning in recent years, there is a growing trend of disseminating and exchanging information via this popular channel. Digital library, Massive open online courses (MOOCs) and distance learning are becoming hot research areas that address issues arisen from the widespread use of the Internet. One of the key vehicles in the drive towards realizing these ideas is to develop cheap and efficient methods for transcribing existing knowledge in the form of paper documents into corresponding electronic form, which is the form that can be processed by today's digital computers and transmitted through the Internet. Mathematical expressions constitute an essential part in most scientific and engineering disciplines. The input of mathematical expressions into computers is often more difficult than that of plain text, because mathematical expressions typically consist of special symbols and Greek letters in addition to English letters and digits. With such a large number of characters and symbols, the commonly used type of keyboard has to be specially modified in order to accommodate all the keys needed. Another method is to make use of some extra keys in the keyboard (e.g., function keys) along with a set of unique key sequences for representing other special symbols. Yet another method is to simply define a set of keywords to represent special characters and symbols, as in LATEX. However, working with specially designed keyboards or keywords requires intensive training and practice. Alternatively, by taking advantage of pen-based computing technologies, one could simply write mathematical expressions on an electronic tablet for the computer to recognize them automatically. In situations where the expressions are already in some printed form, we could just scan in the document for the computer to recognize the expressions directly from the image.

Character recognition, as the most common type of symbol recognition problems, has been an active research area for more than three decades. Structural analysis of two-dimensional patterns also has a long history. However, very few papers had addressed specific problems related to mathematical expression recognition. It is only until recently that more researchers have started to pay attention to this area.

Introduction

The problem of Mathematical expression recognition falls under Text recognition or Optical Character Recognition (OCR) is vast. The complete scope of OCR is vast but fundamentally, it is conversion of printed or handwritten text in images to machine-encoded text. This problem is an important area of research, particularly in data entry from printed records, digitizing printed texts and is also used in machine translation and text mining.

The process of OCR involves several steps including segmentation, feature extraction, and classification. Each of these steps is a field unto itself. A few examples of OCR applications are listed here. The most common for use OCR is the first item; people often wish to convert text documents to some sort of digital representation.

1. People wish to scan in a document and have the text of that document available in a word processor.

2. Recognizing license plate numbers

3. Post Office needs to recognize zip-codes

In the most general case, images contain several words and symbols. If the image contains both handwritten and machine printed text, the problem of Separation of these two also arises. In general, the texts are separated first and then sent for recognition.

Generally, machine printed text represents the original document and handwritten texts add or alter it. This can be widely used in processing of forms and bank cheques.

In the first part of the project we focus on recognition aspect of the task.

Tasks such as OCR fall into a class of problems known as pattern recognition which deals with classifying the given data. Other Examples of Pattern Recognition:

1. Facial feature recognition (airport security) – Is this person a bad-guy?

2. Speech recognition – Translate acoustic waveforms into text.

3. A Submarine wishes to classify underwater sounds – A whale? A friendly ship?

The Classification Process:

(Classification in general for any type of classifier) There are two steps in building a classifier: training and testing. These steps can be broken down further into sub-steps.

1. **Training**

a. Pre-processing – Processes the data so it is in a suitable form

b. Feature extraction – Reduce the amount of data by extracting relevant information. This usually results in a vector of scalar values. (We also need to normalize the features for distance measurements!)

c. Model Estimation – from the finite set of feature vectors, need to estimate a model (usually statistical) for each class of the training data

2. **Testing**

a. Pre-processing

b. Feature extraction – (both same as above)

c. Classification – Compare feature vectors to the various models and find the closest match. One can use a distance measure.

Problem Specification

In a mathematical expression, characters and symbols can be spatially arranged as a complex two-dimensional structure, possibly of different character and symbol sizes. All the characters and symbols, when grouped properly, form an internal hierarchical structure.

Both symbol recognition and structure analysis of two-dimensional patterns have been extensively studied for decades. Mathematical expression recognition, which features both of them as the two major stages of the recognition process, is a good subject for studying the integration of the two areas.

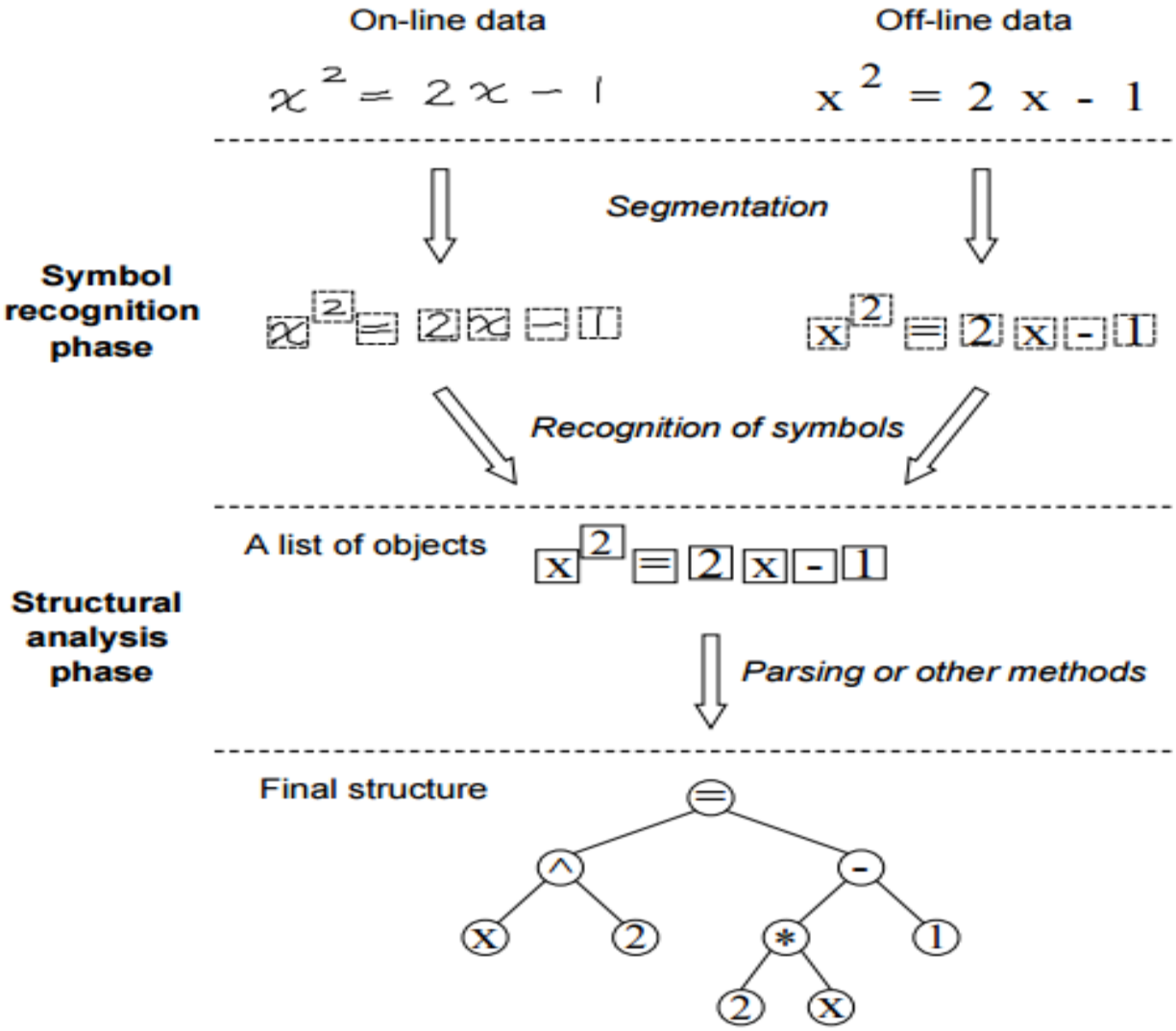
Many symbol recognition techniques work under the assumption that the symbols have already been isolated from each other.

When we write a mathematical expression on a tablet, what we get is a sequence of points. On the other hand, if we scan an expression from a printed document, what we get is a two-dimensional array of pixels. The data in the first case are usually regarded as on-line data while those in the latter case are off-line data. Intuitively, if we are able to segment the data into groups so that each group represents a single symbol, we can then directly apply an existing symbol recognition method to decide its identity. Afterwards, a list of objects with associated attributes (including location, size, and identity) is returned.

After more than three decades of research, many existing symbol recognition techniques are able to achieve quite satisfactory results. However, many of them can only work with isolated symbols. In a mathematical expression, there usually exist multiple symbols. Before we can apply these symbol recognition techniques, we must first segment the individual symbols from the expression properly.

This work aims to address the problems of symbol segmentation and recognition. To this end, we have proposed and implemented our method of segmenting handwritten mathematical symbols and created a dataset from mathematical symbols handwritten by students of IIT Jodhpur. For the recognition part,

we look at different machine learning approaches and feature extraction methods to recognize both on-line and off-line symbols from different datasets.



Overview of an intuitive recognition process

Literature Review

Segmentation of Symbols

A naive way to segment symbols is to put all physically separate components (or called connected components) into groups. However, some characters and symbols, such as `i', `j', and `=', are composed of multiple components. As a result, we have to combine the corresponding components together first before we can recognize the individual characters or symbols correctly. In more complicated situations, some symbols, like `√', usually contain other symbols inside their effective regions. Hence, caution must be taken when segmenting such symbols. Faure and Wang presented a modular system for segmenting handwritten mathematical expressions. Their system has two segmentation modules. The data-driven segmentation module first builds a relation tree of the given expression. In general, projections on the X and Y axes can be used for deciding how to segment the data. However, for symbols like `√' and the fraction line, this approach often fails. A mask removal operation is thus applied to segment these symbols and their embedded symbols before projection operations are carried out for the embedded symbols. Afterwards, the knowledge driven segmentation module attempts to correct the relation tree built by the previous module. For example, it tries to combine different parts together for symbols like `i', `j' and `='. Okamoto et al. proposed to partition a given printed expression into components by recursive horizontal and vertical projection profile cutting. Some additional checking steps are required for symbols which contain separate elements (e.g., `i', `j' and `=') and symbols which contain some other symbols within their regions (e.g., `√'). Ha et al. used the bounding boxes of the symbols as the clue for extracting them from a printed expression. Their method is called “recursive X-Y cut" in which `X' refers to the horizontal cut and `Y' refers to the vertical cut. This method is similar to the projection profile cutting method except for the primitive objects used for projection (one uses pixels and the other uses bounding boxes). Smithies et al. developed a simple progressive grouping algorithm for on-line symbol segmentation. Their system first generates all possible groupings for a small number of strokes (according to a small upper bound). It then looks for the one with the maximal confidence level given by the character recognizer. This algorithm is simple and fast but may sometimes introduce errors that require manual correction. In general, most of the algorithms work quite well.

However, just like other segmentation problems, these algorithms often rely on the use of thresholds. In practice, threshold values cannot be chosen to work well on all possible inputs.

Recognition of Segmented Symbols

After the segmentation step, we have a list of objects with some known attribute values. The only missing values are the identities of symbols. In theory, we can apply any symbol recognition method as long as it is designed for the corresponding data type (i.e., on-line or off-line).

Over many years of research, different approaches have been proposed for symbol recognition, including template matching, structural, neural network and other statistical approaches. Surveys of

these approaches can easily be found in the literature. Here, we do not intend to repeat what have been done. Instead, we will just list some typical systems by category according to the symbol recognition approach used:

1. Template matching approaches:

Several systems, such as Nakayama and Okamoto et al., make use of some traditional template matching methods. Others perform template matching based on different measures, e.g., Fateman et al. and Miller and Viola used Hausdorff distance.

2. Structural approaches:

Not many systems are based on structural approaches. A few exceptions are Belaid and Haton and Chan and Yeung.

3. Statistical approaches:

Quite a number of systems, including Chen and Yin, Fateman et al., Fukuda et al., Lee et al., and Smithies et al., are based on traditional statistical approaches. Others, such as Dimitriadis and Coronado, Ha et al., and Marzinkewitsch use neural networks.

Some methods, such as those based on hidden Markov models (HMMs), have proven to be very effective in the area of speech recognition. Some researchers thus attempted to apply this approach to recognize symbols in mathematical expressions. As mentioned above, the on-line data of a mathematical expression is simply a sequence of points. This is analogous to the case for speech except that speech is a sequence of acoustic signals. Hence, HMM techniques developed for speech recognition can easily be modified for recognizing symbols in on-line mathematical expressions and to achieve simultaneous segmentation and classification. Winkler et al. first generated symbol hypotheses net (SHN) for the handwriting input and then used HMMs to find one or more symbol sequences from the SHN. The final classification of the symbols is done by finding the most probable symbol sequence. By keeping all alternatives for the solution, decision making can be delayed. Such technique is called a soft-decision approach. Sakamoto et al. also used the HMM approach for recognizing characters and symbols in a mathematical expression.

Summary of Symbol Recognition Methods Used

Besides categorizing different systems according to the symbol recognition approach used, we may also group the systems according to the data type required. Table below shows such a categorization.

Data type	Major method	Example
On-line	Structural feature extraction and decision tree classification	Beláid and Haton [5]
	Flexible structural matching	Chan and Yeung [8]
	Feature extraction and nearest neighbor classification	Chen and Yin [11], Fukuda *et al.* [19], Smithies *et al.* [58]
	ART-based neural architecture and elastic matching	Dimitriadis and Coronado [14]
	Hidden Markov model	Winkler *et al.* [32, 43, 64, 66, 67], Sakamoto *et al.* [57]
	Three-layered backpropagation network	Marzinkewitsch [45]
	Traditional template matching	Nakayama [49]
Off-line	Template matching based on Hausdorff distance	Fateman *et al.* [6, 16], Miller and Viola [46]
	Feature extraction and nearest neighbor classification	Fateman *et al.* [17], Lee *et al.* [39–42]
	Feature extraction and classification through neural network approach	Ha *et al.* [22]
	Traditional template matching	Okamoto *et al.* [51, 52]

It is worth noting that the set of symbols used in mathematical expressions can sometimes be very large. Hence, some researchers focus on only a subset of them. For example, Zhao et al. analyzed the structure of 94 commonly used mathematical symbols and discovered that they are all based on 10 basic elements. Several techniques, such as basic element ordering and reduction of number of standard symbols, have been applied to increase the recognition rate.

Part 1: HMM based recognition of Online Mathematical symbols

Producing large and complicated expressions in these days requires a lot of time and mental effort. With the emergence of pen-based electronic devices, such as Smartphones and tablets, people can simply write mathematical expressions on the electronic tablet to let the computer recognize them automatically.

Dataset

We extract features from symbols of a publicly available, ground-truthed corpus of over 4500 handwritten mathematical expressions created by MacLean et.al., called “MathBrush”. These symbols were written by 20 writers. We discard six symbol classes whose samples are less than 50 and the symbol ’dot’, thus getting 53 symbol classes. Table below shows all the 53 classes of symbols which can be recognized in our system.

0	1	2	3	4	5	6	7
8	9	a	A	b	B	c	C
d	e	F	i	j	k	x	y
z	X	tan	log	lim	lt	cos	+
(	)	!	=	-	π	ϕ	n
θ	α	β	γ	$\rightarrow$	∞	$\surd$	$\neq$
$\sum$	$\geq$	$\leq$	$\int$	sin			

The dataset is unbalanced, and different symbol have different number of samples. For each class of symbol, we use 90% samples as the training set and the other 10% as the testing set. There are 10359 samples in the training set and 1151 samples in the testing set.

Methodology

A. Preprocessing

The preprocessing procedure consists of four steps: duplicate point filtering, size normalization, smoothing and resampling. These steps reduce noise and unuseful information for classification.

Duplicate point filtering:

Duplicate point is the point that has the same (x; y) coordinates as the previous point and cannot give any useful information for classification.

Size normalization:

The class of a symbol is independent of its size, therefore size normalization is needed to eliminate the variation of size. It is achieved by transforming the y coordinate's range to be [0; 1] while preserving the width-height aspect ratio.

Smoothing:

Smoothing is used to reduce the noise information caused by the digital pen's jitter. Except the first point and the last of every stroke, the other points' coordinates are replaced by the average of the coordinates of current point, the previous point and the following point.

Resampling:

The original points are recorded equidistantly in time but not in space. In order to remove the influence of writing velocity, we resample each symbol to 30 points along the original trajectory with equal distance between the consecutive points.

B. Feature Extraction

Liwicki et al. applied a sequential forward search on a feature set in order to discover which features are significant for handwriting recognition. A Hidden Markov Model and a bidirectional long short-term memory network (BLSTM) based recognizer were used as recognition engines. They applied many operations to reduce noise and normalize the skew, slant, width and height before feature extraction. There are 25 features in the feature set: (1) pen-up/down, (2) hat-feature, (3) speed, (4) normalized x-coordinate, (5) normalized y-coordinate, (6,7) cosine and sine of writing direction, (8,9) cosine and sine of curvature, (10-18) context map, (19) vicinity aspect, (20) vicinity curliness, (21) vicinity linearity, (22,23) cosine and sine of vicinity slope, (24) ascenders, and (25) descenders.

The experiment results showed that the recognition rate with only five features approaches the recognition rate using all the 25 features. The experiment results also show that the first five iterations of the sequential forward search algorithm with HMM based classifier and BLSTM based classifier have selected the same best five features.

In addition, in the first five iterations of the sequential forward search algorithm with HMM based classifier, the ranking of the first five features does not change. It can be concluded that the five features are very stable and contain more important information than other features for the classification.

The best five features are the cosine of the slope, the normalized y-coordinate, the density in the center of the context maps, the pen-up/down information, and the sine of the curvature.

We use all four online features among the best five features: the cosine of the slope, the normalized y-coordinate, the pen-up/down information, and the sine of the curvature.

A 4-dimensional feature vector is computed for each point of the sample. Because the number of the features is small, we can get a more efficient classifier, in terms of computation and storage.

1) Pen-up/down:

A binary feature denoting whether the digital pen has contact with the electronic tablet or not at time t.

Now, we use information from this feature to get a feature called Normalized distance to stroke edge (NDTSE): in order to add the location information to the pen-up/down feature, we take the distances to the beginning and the end of the stroke into account and replace the pen-up/down feature with NDTSE. The new feature can be computed as:

For actual stroke, it is $1 - \frac{|d_e - d_b|}{l_s}$

For interpolated stroke, it is $-\left(1 - \frac{|d_e - d_b|}{l_s}\right)$

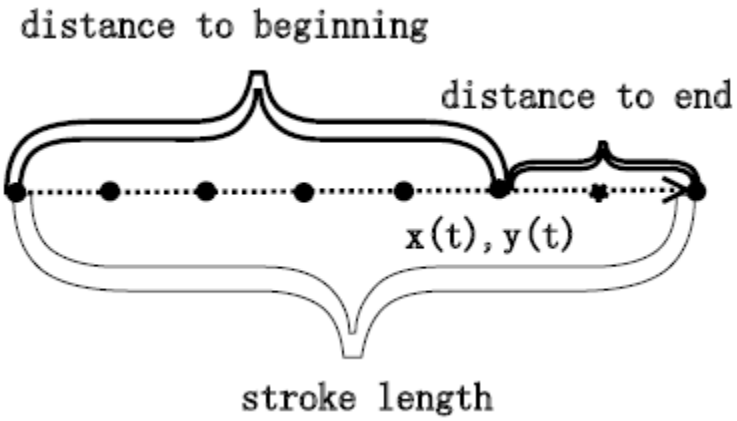
Here:

ls represents the length of the stroke s which the current point x(t); y(t) belongs to;

de represents the distance between the current point and the last point of s;

db represents the distance between the current point and the first point of s.

Actual stroke is the visible stroke, while interpolated stroke is the hidden parts of the trajectory, where the digital pen does not contact with the electronic tablet. For the point belongs to actual stroke, NDTSE is nonnegative; for the point belongs to interpolated stroke, NDTSE is nonpositive. Following figure visualizes the new feature.



2) Normalized y-coordinate:

The vertical position after size normalization.

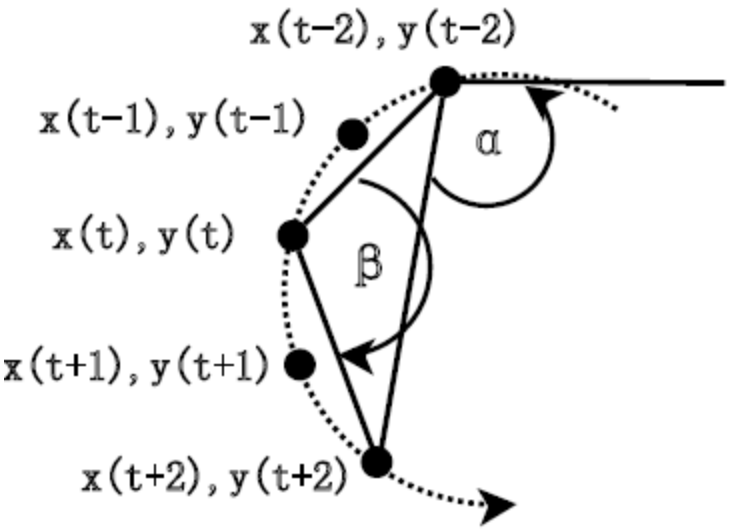
3) Vicinity slope(α):

The vicinity slope of the current point (x(t); y(t)) is represented by the cosine of the angle between the straight line connecting the point (x(t-2); y(t-2)) and the point (x(t+2); y(t+2)) and the horizontal across the point (x(t-2); y(t-2)).

2) Curvature(β):

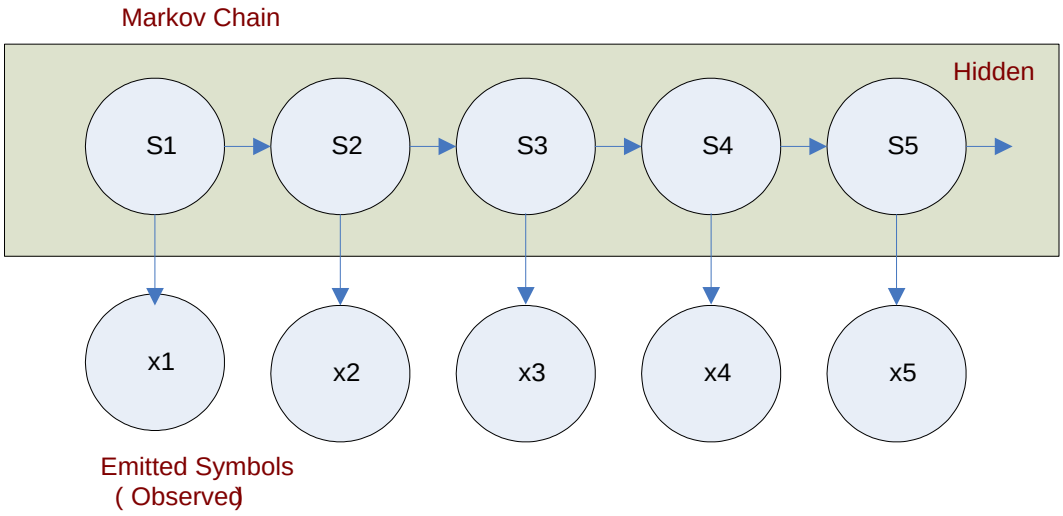
The curvature of the current point (x(t); y(t)) is represented by cosine and sine of the angle between the straight line joining point (x(t - 2); y(t- 2)) and point (x(t); y(t)) and the straight line joining point (x(t); y(t)) and point (x(t + 2); y(t + 2)).

Following figure represents the slope and curvature:



C. HMM classifier

An HMM is a statistical model of a Markov process with hidden states. One use of an HMM is to determine the relationship of the hidden states to the observations, which depends on the associated probabilities. The graphic representation of the HMM is illustrated in figure below.



The states of the Markov chain are hidden. The outputs from the Markov chain are observable. The Hidden Markov model is a statistical model for a sequence of observation items. There are three different sub-problems that are very useful in many application areas. Those three areas are:

- The evaluation problem: The evaluation problem is to find the probabilities of the observation given a sequence and a model. The solution of this problem measures how well a model matches the observation sequence.
- The decoding problem: given the model and a sequence, what is the optimal state sequence? The solution to this problem is useful for solving word segmentation problem such as classifying compound words.
    - The learning problem: The learning problem is the most interesting of all three problems. If there is existing models, we can use this technique to find the model for a sequence and re-apply the sequence to the model for the learning and decoding problem. This is extremely useful. It is also the problem that our experiments are focused on. The technique is to apply the forward-backward algorithm to this problem and use the Baum-Welch Algorithm to refine the model parameters.

In this work, we will be looking at the third problem. Our intention is to find the models that can best describe our feature vector sequences. We will give a brief overview of the mathematics behind this beautiful technique.

HMM definition

A Hidden Markov Model (HMM) is a probabilistic function of a Markov property. The term "hidden" indicates that the system is a doubly stochastic process where the state $q_1^t = \{q_1, \cdots\cdots, q_t\}$ of the Markov chain is not directly observed (hence the term, "hidden"), but it is implicitly defined by a sequence $y_1^t = \{y_1, \cdots\cdots, y_t\}$ of the observed data that does not necessarily exhibit the Markov property.

A HMM is defined by the following conditional probabilities.

1. Given q_t , the distribution of y_t is independent of every other variable,

$$P\left(y_t \middle| q_1^t, y_1^{t-1}\right) = P\left(y_t \middle| q_t\right)$$

2. y_t is unable to affect q_t given the past.

$$P\left(q_{t+1} \middle| q_1^t, y_1^t\right) = P\left(q_{t+1} \middle| q_t\right)$$

Without the HMM assumptions, we are unable to calculate the conditional probability since it would be intractable if all the history is required to be considered.

The probability of the observed sequence, written as $P\left(y_1^T\right)$, can be calculated by finding the joint probability of the observation sequence and the state sequence $P\left(y_1^T, q_1^T\right)$. The term $P\left(y_1^T, q_1^T\right)$ can be recursively factored using conditional probability and chain rules.

$$\begin{aligned} P\left(y_1^T, q_1^T\right) &= P\left(y_T, q_T \middle| y_1^{T-1}, q_1^{T-1}\right) P\left(y_1^{T-1}, q_1^{T-1}\right) \\ &= P\left(y_T \middle| q_T, y_1^{T-1}, q_1^{T-1}\right) P\left(q_T \middle| y_1^{T-1}, q_1^{T-1}\right) P\left(y_1^{T-1}, q_1^{T-1}\right) \\ &= P\left(y_T \middle| q_T\right) P\left(q_T \middle| q_{T-1}\right) P\left(y_1^{T-1}, q_1^{T-1}\right) \\ &= P(q_1) \prod_{t=2}^{T} P\left(q_t \middle| q_{t-1}\right) \prod_{t=1}^{T} P\left(y_t \middle| q_t\right) \end{aligned}$$

where

$P\left(q_1\right)$ is the initial state probability distribution of *q* at time 1

$P\left(q_t \middle| q_{t-1}\right)$ is the probability of *q* at time *t* given *q* at time *t+1*

$P\left(y_t \middle| q_t\right)$ is the emission probability

We can get the desired probability by marginalizing (summing) over random variables $q_{1...T}$:

$$P\left(y_1^T\right) = \sum_{q_{1...T}} P\left(y_1^T, q_1^T\right)$$

Forward-Backward algorithm

The direct approach is computationally infeasible since it has an exponential order of complexity, $(2T-1)N^T$. Instead, we can calculate the joint probability using an alternative method called the forward-backward algorithm. The forward-backward algorithm is a multi-pass algorithm that reduces the complexity of the computation.

Forward recursion

The forward recursion calculates, $P\left(y_1^t, q_t\right)$, the probability of an observed partial sequence y_1^t for a given state q_t . We can rewrite this joint probability as a conditional probability in the product form: $P\left(y_1^t, q_t\right) = P\left(y_t \left| y_1^{t-1}, q_t\right.\right) P\left(y_1^{t-1}, q_t\right)$. According to the HMM assumption, the term $P\left(y_t \left| y_1^{t-1}, q_t\right.\right)$ can be reduced to $P\left(y_t \left| q_t\right.\right)$. Thus, we need to calculate $P\left(y_1^t, q_1^{t-1}\right)$ to complete the equation.

$$\begin{aligned} P\left(y_1^t, q_t, q_{t-1}\right) &= P\left(q_t \left| q_{t-1}, y_1^{t-1}\right.\right) P\left(y_1^{t-1}, q_{t-1}\right) \\ &= P\left(q_t \left| q_{t-1}\right.\right) P\left(y_1^{t-1}, q_{t-1}\right) \end{aligned}$$

$$\begin{aligned} P\left(y_1^{t-1}, q_t\right) &= \sum_{q_{t-1}} P\left(y_1^t, q_t, q_{t-1}\right) \\ &= \sum_{q_{t-1}} P\left(q_t \left| q_{t-1}\right.\right) P\left(y_1^{t-1}, q_{t-1}\right) \end{aligned}$$

Hence, we get the following equation.

$$P\left(y_1^t, q_t\right) = P\left(y_t \left| q_t\right.\right) \sum_{q_{t-1}} P\left(q_t \left| q_{t-1}\right.\right) P\left(y_1^{t-1}, q_{t-1}\right)$$

This equation exhibits a recurrence relation so that $P\left(y_1^t, q_t\right)$ can be calculated recursively. We define $\alpha_q(t) = P\left(y_1^t, q\right)$ then the above equation can be expressed as

$$\alpha_q(t) = P\left(y_t \middle| Q_t = q\right) \sum_r \left(Q_t = q \middle| Q_{t-1} = r\right) \alpha_r(t-1)$$

where Q_t is the state space at time *t*

Backward recursion

Once we complete the forward phase, we still need to calculate the backward phase in the algorithm. The backward phase calculates the partial probability from time *t+1* to the end of the sequence, given q_t.

$$\begin{aligned} P\left(y_{t+1}^T \middle| q_t\right) &= \sum_{q_{t+1}} P\left(q_{t+1}, y_{t+1}, y_{t+2}^T \middle| q_t\right) \\ &= \sum_{q_{t+1}} P\left(y_{t+2}^T \middle| q_{t+1}, y_{t+1}, q_t\right) P\left(y_{t+1} \middle| q_{t+1}, q_t\right) P\left(q_{t+1}, q_t\right) \\ &= \sum_{q_{t+1}} P\left(y_{t+2}^T \middle| q_{t+1}\right) P\left(y_{t+1} \middle| q_{t+1}\right) P\left(q_{t+1}, q_t\right) \end{aligned}$$

In the backward phase, $P\left(y_{t+1}^T \middle| q_t\right)$ can only be calculated once we have the information about $P\left(y_{t+2}^T \middle| q_{t+1}\right)$. This is also called backward recursion.

We can define $\beta_q(t) = P\left(y_{t+1}^T \middle| Q_t = q\right)$, and the above equation can be expressed as

$$\beta_q(t) = \sum_r \beta_r(t+1) P\left(y_{t+1} \middle| Q_{t+1} = r\right) P\left(Q_{t+1} = r \middle| Q_t = q\right)$$

where Q_t is the state at time *t*

The forward-backward recursion gives us the essential information to calculate the probability of the observed sequence $P\left(y_1^T\right)$.

$$
\begin{aligned}
P\left(y_1^T\right) &= \sum_{q_t} P\left(q_t, y_1^t, y_{t+1}^T\right) \\
&= \sum_{q_t} P\left(y_{t+1}^T \middle| q_t, y_1^t\right) P\left(q_t, y_1^t\right) \\
&= \sum_{q_t} P\left(y_{t+1}^T \middle| q_t\right) P\left(q_t, y_1^t\right) \\
&= \sum_{q_t} \beta_{q_t}(t)\alpha_{q_t}(t)
\end{aligned}
$$

Choosing best state sequence

We want to find the most likely state sequence q_1^T corresponding to a given observation sequence y_1^T. Since our interest is in the overall model performance rather than finding a specific sequence, we will use an approach to maximize the expected number of states for our HMMs.

The state posterior probability, $P\left(q_t \middle| y_1^T\right)$, is the probability of being in a certain state at time t, given the observation sequence y_1^T. It can be expressed using the variables that we defined from previous forward and backward processes.

The probability, $P\left(q_t \middle| y_1^T\right)$, is simply the product of forward-backward variables and normalized by the joint distribution of the observation sequences. Hence we express it as the following equation:

$$
P\left(q_t \middle| y_1^T\right) = \frac{P\left(q_t, y_1^T\right)}{P\left(y_1^T\right)} = \frac{P\left(y_1^t \middle| q_t\right) P\left(q_t\right) P\left(y_{t+1}^T \middle| q_t\right)}{P\left(y_1^T\right)} = \frac{P\left(y_1^t, q_t\right) P\left(y_{t+1}^T, \middle| q_t\right)}{P\left(y_1^T\right)}
$$

Since $P\left(q_t \middle| y_1^T\right)$ is normalized by the joint distribution of the observation sequences. $\sum P\left(q_t \middle| y_1^T\right) = 1.$

The most likely state is measured by maximizing $P\left(q_t \middle| y_1^T\right)$ for q_t.

Parameter estimation

The transition posterior probability, $P\left(q_t, q_{t-1} \middle| y_1^T\right)$, can be expressed as :

$$P\left(q_t, q_{t-1} \middle| y_1^T\right) = \frac{P\left(y_t \middle| q_t\right) P\left(y_1^{t-1}, q_{t-1}\right) P\left(y_{t+1}^T \middle| q_t\right) P\left(q_t \middle| q_{t-1}\right)}{P\left(y_1^T\right)}$$

Note that if we marginalize the probability of $P\left(q_t, q_{t-1} \middle| y_1^T\right)$ over all possible state q_{t-1}, the result is the probability $P\left(q_t \middle| y_1^T\right)$. Once we obtain the state posterior, we can calculate the expected number of time that a certain state q_t is visited by simply summing over the time index t: $\sum_{t=1}^{T-1} P\left(q_t \middle| y_1^T\right)$

To calculate the expected number of time that a transition from state i transits to state j, we sum all the $P\left(q_t, q_{t-1} \middle| y_1^T\right)$ over time index t, $\sum_{t=1}^{T-1} P\left(q_t = i, q_{t-1} = j \middle| y_1^T\right)$

Hence, we can re-estimate the HMM parameter using the formulas that we describe as follows.

- The re-estimated initial state probabilities are simply the expected frequencies of the states at time $t = 1.$

- The re-estimated transition probabilities are the expected numbers of transition from q_t to q_{t-1} over the expected number of transitions

$$\frac{\sum_{t=1}^{T-1} P\left(q_t = i, q_{t-1} = j \middle| y_1^T\right)}{\sum_{t=1}^{T-1} P\left(q_t \middle| y_1^T\right)}$$

- The re-estimated emission probabilities are the expected number of time that a certain state i is visited for the specific observation symbols over the expected number of time in a particular state.

$$\frac{\sum_{t=1}^{T-1}{}_{Y=y} P\left(q_t \middle| y_1^T\right)}{\sum_{t=1}^{T-1} P\left(q_t \middle| y_1^T\right)}$$

A HMM is specified by the parameter set (A;B; π). A denotes the state transition probability matrix; B denotes the observation probability distribution; π is the initial state distribution.

Each written symbol can be represented by a sequence of feature vectors O, defined as

O = O1;O2;……. ;OT ;

where Ot is the feature vector observed at time t. The goal of the HMM classifier is to find the probability that a specific class is the most likely to occur given a sequence of observations.

1) Model Selection:

There is no theoretically optimal method to choose the type of model (ergodic or left to right), the model size (number of states) and observation probability distribution (discrete or continuous, single or multi-mixture) for an HMM. The type of model, the model size and the observation probability distribution are determined empirically.

In our HMMs, we use the linear topology. For each state, only the transition to itself or the next state is permitted. The observation probability for a given feature vector is determined by Gaussian Mixture Models and the covariance matrix of the mixture component is diagonal.

To choose the model size and the number of Gaussian components per state, I did experiments on the ten digits extracted from the corpus of handwritten mathematical expressions to find the effect of number of states and number of Gaussians on the recognition rate. The experiment results are shown in table below.

Model Size	top-1
3 states 4 Gaussians	0.960
4 states 4 Gaussians	0.965
5 states 4 Gaussians	0.966
6 states 4 Gaussians	0.968
7 states 4 Gaussians	0.969
6 states 2 Gaussians	0.960
6 states 3 Gaussians	0.963
6 states 5 Gaussians	0.974
6 states 6 Gaussians	0.968

When the number of states is fixed to be 6, model with 5 Gaussians per state gets the best performance. Therefore, each model has six states and each state contains five Gaussians in our system.

2) Initialization:

Theoretically speaking, the re-estimation process of Baum-Welch algorithm can assign values to the HMM's parameters which can make the likelihood function to get a local maximum. But there is no

straightforward way to choose good initial estimates of the HMM parameters to guarantee that the local maximum is the global maximum or a strong local maximum of the likelihood function. In most cases, either random or uniform initial estimates of the initial state distribution and state transition probability matrix A is enough with Baum-Welch algorithm for producing useful re-estimates of these parameters. But the re-estimates of the Gaussian parameters are very sensitive to the initial estimates. Therefore good initial estimates of Gaussian parameters are necessary. In this work, I set the initial state distribution to be

$\pi = [1; 0; 0; 0; 0; 0]$

and keep it fixed during the training process. That means the first feature vector out of a sequence is fixed to the first state. But we don't fix the last feature vector out of a sequence to the last state. Discrete uniform distribution is used to give the initial state transition probability matrix A. I use a variant of segmental K-means algorithm to get the initial parameters of observation probability distribution B. We first assign random values to the Gaussian parameters. Then over five iterations, we use the Viterbi algorithm to get the optimal path, having to terminate at the final state, of all observation sequences and segment the feature vector of each point according to the optimal path into the six states. After each state gets the set of the feature vectors that are assigned to it in the current and all previous iterations, K-means algorithm is used to cluster the observations into five clusters and update the Gaussian parameters of each state. But segmental K-means segments all training sequence according to the optimal path given by the Viterbi algorithm. Each state just can get the observations that occur within it in the current iteration.

3) Training:

There are a number of methods for the HMM training, here I use the Baum-Welch algorithm. The Baum-Welch algorithm.

4) Recognition:

In the recognition phase, all HMMs are used with the Forward algorithm to calculate the probability of the observation sequence, O = O1O2…….OT , given the model λ, P(O| λ). The symbol with the maximal class conditional probability will be selected as the class label.

Results

On applying the algorithm on the 1151 randomly selected samples to form test dataset, we get a top- 1 recognition rate of 93.39% and top-5 recognition rate of 98.95%.

Part 2: Creating Dataset of Handwritten Mathematical symbols

To analyze the previous algorithm and further algorithms on offline handwritten symbols we thought of creating a dataset of handwritten mathematical symbols.

Methodology

To get a reliable dataset with sufficient samples to include varying styles of writing symbols and handwriting patterns, we collected samples of various handwritten symbols of students of IIT Jodhpur. In total 43 symbol classes were collected on separate sheets of paper for each class and then segmentation of individual symbols was carried out. The following table lists the symbols collected.

0	1	2	3	4
5	6	7	8	9
a	A	b	B	c
d	e	F	i	j
k	x	y	z	n
cos	tan	(	)	!
+	X	=	α	β
γ	-	$\rightarrow$	$\surd$	ϕ
π	θ	∞		

The dataset was collected from samples of mathematical symbols written by 19 people. The images were manually scanned at 600 dpi. A sample of scanned page is shown below:


tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan.
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan
tan tan tan tan tan

Segmentation

To segment out the individual characters from the scanned pages an adaptive filtering algorithm was used.

1) Adaptive filtering:

The RGB image was loaded and then we applied adaptive Wiener filter to each of the three channels to remove the noise and ten recombined the three channels. Wiener deconvolution deblurs the image when the point-spread functions and noise levels are unknown. The filter uses a pixelwise adaptive Wiener method based on statistics estimated from a local neighborhood of each pixel.

2) Otsu's segmentation:

The image was converted to grayscale and then Otsu's algorithm was applied to choose the threshold to minimize the intraclass variance of the black and white pixel. The algorithm uses histogram of grayscale image to get optimal threshold. The pixles above and below the threshold were classified as 1's and 0's to get a binary image

3) Smoothing:

The binary image obtained has too many connected components due to residual noise and false positives in segmentation. A predefined kernel of averaging filter is convolved with the binary image to reduce the number of connected components and smooth the image

4) Removal of small noise pixels:

Connected components were found out in the image by the following algorithm:

a. Search for the next unlabeled pixel, p.
b. Use a flood-fill algorithm to label all the pixels in the connected component containing p.
c. Repeat steps a and b until all the pixels are labelled.

After getting the connected components the area of each connected component was determined and then objects smaller then the threshold are were removed and filled with background pixels.

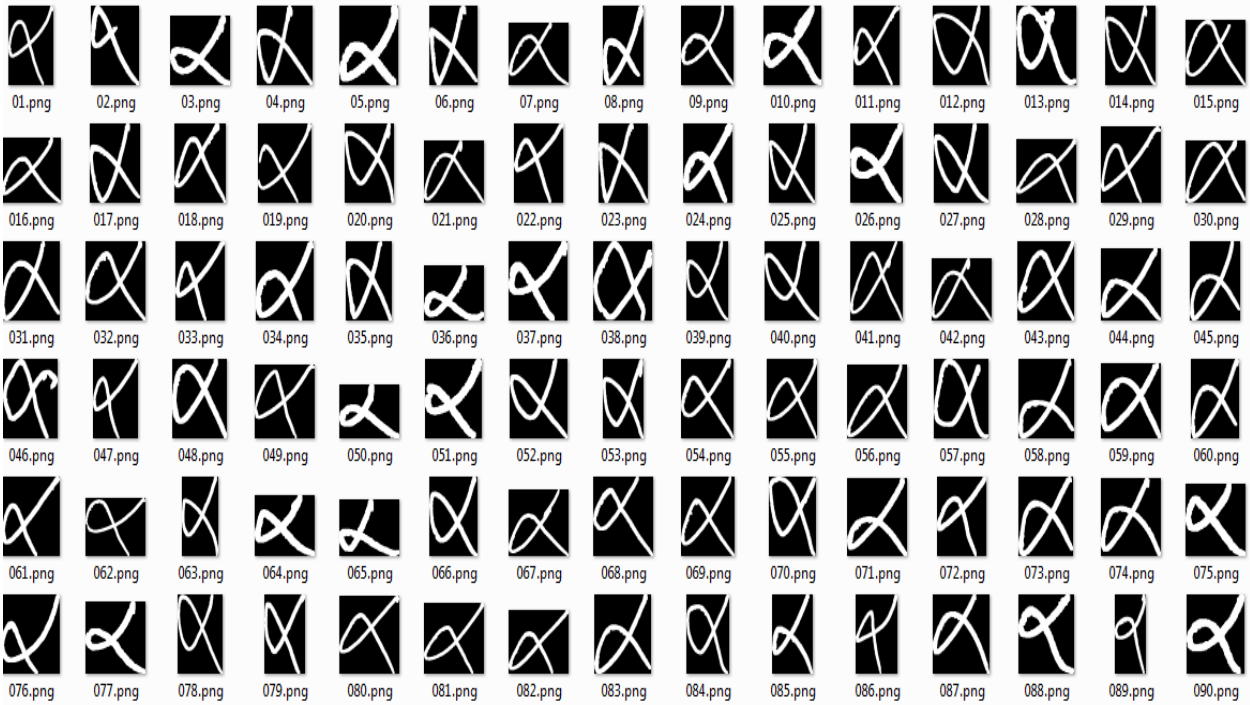
After this step we get a very good binary image relatively free from any noise.

5) Boxing:

Again the connected components of the image are found and bounding boxes constructed over the extrema of each component. Then, individual boxes are segmented out and stored in appropriate class label.

Results

The resulting symbols were segmented properly and were free from noise. The binary images of each instance of a symbol was stored under suitable label. The only instance of the algorithm giving out images with extra component was with ‘j’ and ‘!’ due to the problems discussed previously but the method worked well for ‘=’. The solution to the problem in case of ‘j’ and ‘!’ was using a different type of smoothing kernel and different shape to propagate the regions for finding connected components. The following images are an example of the resulting binary images obtained.



In total 43 symbol classes were collected with approximately 90 instances of each symbol. Binary images of each symbol instances are noise free and good for learning algorithms.

Part 3: Offline recognition

Offline recognition implies recognizing the image of already handwritten symbols. Here the temporal information regarding the symbol structure formation is lost making this a challenging task.

Methodology

Approach 1:

The first approach we used was to extract the feature from existing offline database and classify using our pre-trained HMM to see the results.

The MNIST dataset provides a training set of 60,000 handwritten digits and a validation set of 10,000 handwritten digits. The images have a size of 28×28 pixels. We extracted the image from this dataset.

After extraction, the x and y coordinates of the foreground pixels of digits were found out to get a set of coordinates similar to the online case.

A. Preprocessing

The preprocessing procedure is similar to the previous method for online case but consists of four steps: duplicate point filtering, size normalization, smoothing and resampling. These steps reduce noise and unuseful information for classification. The former three algorithms were unchanged but the latter was modified to just resample the points in image to 30 equidistant points.

B. Feature Extraction

We use all four online features from the previous case and applied the algorithms on the pre-processed coordinate set. The pen-up/down information is unavailable in this case so this feature was replaced by normalized x coordinate of image to represent spatial structure of the symbol. These features were then concatenated to get a 30x4- feature matrix for each sample.

Results

A testset of 1000 samples from MNIST was constructed after feature extraction and the trained HMM applied on it. The results got were very poor with accuracy of less than 10%. This was primarily due to the fact that we used HMM trained on online information to the case of offline information where the feature themselves had change and the mismatch of the feature space lead to the poor results. Also, the coordinates in the online case represent a symbol of single point thickness while the offline digits have variable thickness and temporal information is lost.

Further processing and re-evaluation

Then, the images original images of MNIST were further processed.

Skeletonization:

Morphological operations were applied on each binary image of the dataset to remove pixels from the boundary of digits without allowing the objects to break apart to get a 1-pixel spine without changing the Euler number.

Training:

The skeletonized images were then again preprocessed as described previously and the HMM was trained using the feature vectors of these images.

Testing:

The model was trained for 1000 images from the dataset and tested on a set of 100 images.

Results and discussion:

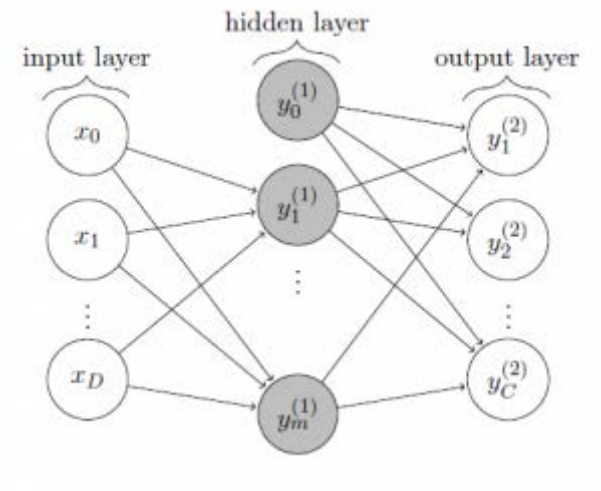
The results were varying in accuracy but on average were between 39 % to 56% in terms of accuracy. The primary reason for this is that HMM works well for sequential inputs due to its inherent structure and definition but due to the offline nature of features the temporal information was lost and so HMM couldn't perform well. Using the newly created dataset for training too didn't give any improvement in accuracy.

Approach 2: Neural network for handwritten digit recognition

The MNIST dataset provides a training set of 60,000 handwritten digits and a validation set of 10,000 handwritten digits. The images have a size of 28×28 pixels. We want to train a two-layer perceptron to recognize handwritten digits, i.e. given a new 28×28 pixels image, the goal is to decide which digit it represents.

Methodology

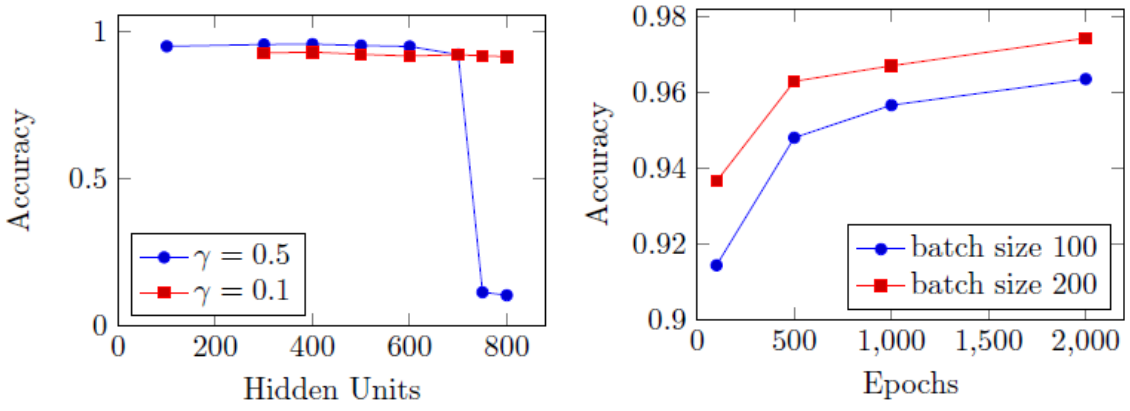
I implemented a 2 layer perceptron in MATLAB to the problem of recognizing handwritten digits. For this purpose, the two-layer perceptron consists of *28·28=784* input units, a variable number of hidden units and *10* output units. The general case of a two-layer perceptron with ***D*** input units, ***m*** hidden units and ***C*** output units is shown in figure



The network is trained using a stochastic variant of mini-batch training, the sum-of-squared error function and the error backpropagation algorithm. The method returns the weights of the hidden layer and the output layer after training as well as the normalized sum-of-squared error after the last iteration.

First the MNIST dataset needs is loaded using the methods `loadMNISTImages` and `loadMNISTLaels`. The labels are provided as vector where the ***i***th entry contains the digit represented by the ***i***th image. We transform the labels to form a 10×N matrix, where N is the number of training images, such that the ***i***th entry of the nth column vector is 1 iff the nth training image represents the digit ***i***–1.
The network is trained using the logistic sigmoid activation function, a fixed batch size and a fixed number of iterations. The training method returns the weights of the hidden layer and the output layer as well as the normalized sum-of-squared error after the last iteration. Then, the numbers of classification errors on the validation set are counted.

Results



(a) 500 epochs with batch size 100.

(b) 500 epochs with learning rate $\gamma = 0.5$.

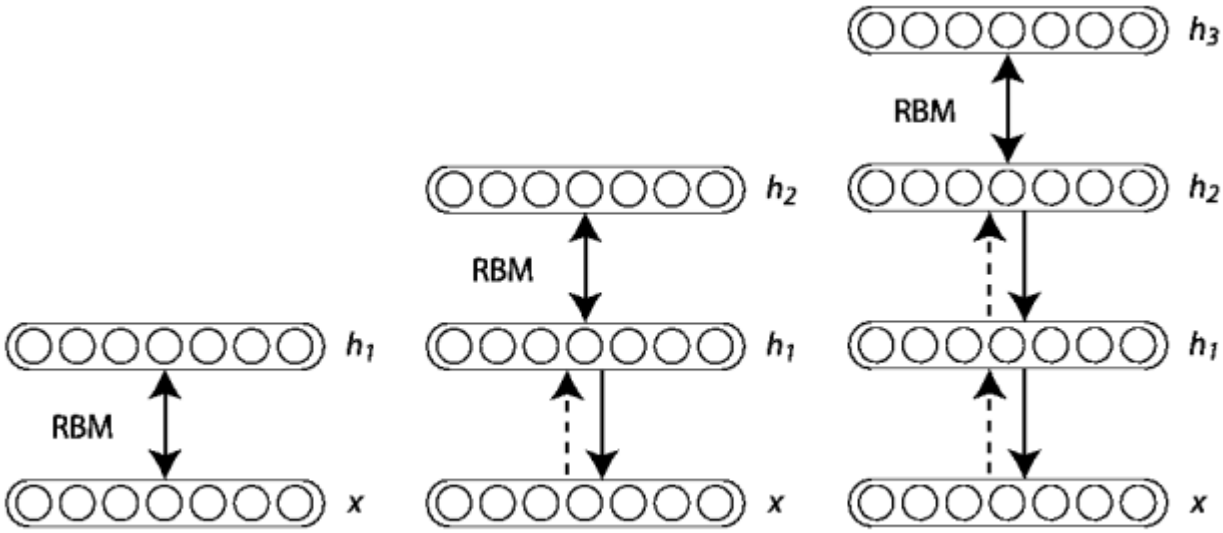
Figure (a) plots the achieved accuracy depending on the number of hidden units for a fixed learning rate of 0.5. For 750 or more units we need to adjust the learning rate in order to ensure convergence. But this may result in slow convergence such that we get worse results for few hidden units. Figure (b) shows the impact of increasing the number of epochs. As we would expect, the accuracy increases with rising training set.

Approach 3: Deep belief networks

Deep learning is basically about hierarchies and abstractions. These hierarchies are controlled by the number of layers in the network along with the number of nodes per layer. Adjusting the number of layers and nodes per layer can be used to provide varying levels of abstraction.

In general, the goal of deep learning is to take low level inputs (feature vectors) and then construct higher and higher level abstract "concepts" through the composition of layers. The assumption here is that the data follows some sort of underlying pattern generated by many interactions between different nodes on many different layers of the network.

Following figure shows an example of training a Deep Belief Network by constructing multiple Restricted Boltzmann Machines stacked on top of each other. Each layer consists of multiple nodes which feed into the next layer.



This review is by not complete and exhaustive and in many cases details have been greatly simplified. This work is meant to be a gentle introduction to DBNs and getting our hands on implementing them.

Methodology

Deep Belief Networks consist of a hierarchy of unsupervised Restricted Boltzmann Machines (RBMs), output of each RBM serves as input to the next. The breakthrough paper in 2006 of Hinton et al. demonstrated that each of the hidden layers in a neural net can be treated as an unsupervised Restricted Boltzmann Machine with a supervised back-propagation step for fine-tuning. Also, the RBMs can be trained greedily and thus were feasible as highly scalable and efficient machine learning models.

This efficiency is further demonstrated in the recent years where Deep Nets have been trained on GPUs rather than CPUs leading to a reduction of training time by over an order of magnitude.

Input Layer:

The first layer is a type of visible layer called an input layer. This layer contains an input node for each of the entries in our feature vector. For example, in the MNIST dataset each image is 28 x 28 pixels. If we use the raw pixel intensities for the images, our feature vector would be of length 28 x 28 = 784, thus there would be 784 nodes in the input layer.

Hidden Layer :

From there, these nodes connect to a series of hidden layers. Simple speaking, each hidden layer is an unsupervised Restricted Boltzmann Machine where the output of each RBM in the hidden layer sequence is used as input to the next. The final hidden layer then connects to an output layer.

Output Layer:

This layer contains the output probabilities for each class label. Here, in MNIST dataset we have 10 possible class labels (one for each of the digits 1-9). The output node that produces the largest probability is chosen as the overall classification.

This work is based on the on nolearn package in python. We use 33% of the data for testing, while the remaining 67% will be utilized for training our Deep Belief Network.

The structure of network is represented as a list. The first entry in the list is the number of nodes in input layer. Input layer feeds forward into second entry in the list, a hidden layer. This hidden layer will be represented as RBM with 300 nodes. The output of the 300 node hidden layer will be fed into the output layer, which consists of an output for each of the class labels.

Then we call the predict method of the network which takes testing data and makes predictions regarding which digit each image contains.

Results

Deep Belief Network is trained over 10 epochs (iterations over the training data). At each iteration, the loss function is minimized and the error on the training set is lower.

symbol	precision	recall	F1 score
0	0.98	0.99	0.99
1	0.99	0.98	0.99
2	0.98	0.98	0.98

3	0.97	0.98	0.97
4	0.98	0.98	0.98
5	0.98	0.97	0.98
6	0.98	0.98	0.98
7	0.99	0.98	0.98
8	0.97	0.97	0.97
9	0.97	0.97	0.97
Average	0.98	0.98	0.98

Here we see that we have obtained 98% accuracy on testing set. The “1” and “7” digits was accurately classified 99% of the time.

Discussion and Future Work

The goal of the thesis is to explore Mathematical Symbol Recognition techniques in images from variety of sources, and develop algorithms for separation and recognition. The algorithms for recognition have been analyzed and demonstrated to recognize text of various styles and settings, and the goal is to obtain successful recognition with minimal error.

The dataset of handwritten mathematical symbols, developed after segmenting the symbols is made as noise free as possible and would serve as a good starting point for IITJ's own dataset repository. Moreover it would help if we have more left-handed samples in the dataset to get more variation in writing styles, though this topic wasn't covered in this work and needs to be explored further.

The results obtained for the online approach (HMMs) show very promising results and this makes HMMs very suitable for application in online setting.

However, in the second case of offline recognition, the performance of HMMs was poor. This arises primarily due to the fact that HMMs couldn't make use of the advantage of their special structure for temporal processes as temporal information is lost in offline case. On the other hand, Neural Networks and Deep belief networks approach were successfully able to recognize to good extent. The output was satisfactory, although not wholly free from error at all times.

There have been new approaches proposed, based on LSTM and RNNs they show promising results in the task. Sparse coding, CRFs, wavelets, component analysis and ensemble based methods have also shown promising results in offline recognition of symbols. Also, I am working on implementing different feature vectors which can somehow take advantage of the fact that in Indo-European languages one writes from left-to-right and partially make up for the loss of temporal information. Also, features based on projection profiles don't work well for skeletonized images, in non-skeletonized case we can make use of projection profile as feature to HMM as it also carries some temporal information. Another task is to explore use of line segments of symbols and their orientations to get better recognition by HMMs. Also sliding window approaches and zoning have also been implemented in literature for case of offline recognition using HMMs and as a logical step, can be implemented to get features which somehow provide partial temporal information.

However, there is a large room of improvement in the quality of recognition. Firstly, the preprocessing needs to be improved to reduce a lot of noise and compensate for orientations, especially in the natural images and videos from webcam where the symbols are written in different orientations. Lastly, the training data set needs to be bigger in order to improve the training of the classifier (especially NNs) which no doubt would improve the results.

Note: I would like to thank Dr. Harit for his valuable mentoring and friends at IITJ for helping in dataset

References

1. S. MacLean, G. Labahn, E. Lank, M. Marzouk, and D. Tausky, “Grammar-based techniques for creating groundtruthed sketch corpora,” International Journal on Document Analysis and Recognition, pp. 1–21, May 2010
2. T.-M.-T. Do and T. Artieres, “Maximum margin training of Gaussian HMMs for handwriting recognition,” In Proc. International Conference on Document Analysis and Recognition
3. Y. Zhang, G. Shi, and J. Yang, “HMM-based online recognition of handwritten chemical symbols,” In Proc. International Conference on Document Analysis and Recognition
4. U. Garain and B. Chaudhuri, “Recognition of online handwritten mathematical expressions,” IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics
5. Juang, B.H., Rabiner, L.: The segmental k-means algorithm for estimating parameters of hidden markov models. IEEE Trans. on ASSP
6. M. Pastor, A. Toselli, and E. Vidal, “Writing speed normalization for on-line handwritten text recognition,” In Proc. International Conference on Document Analysis and Recognition
7. M. Liwicki and H. Bunke, “Feature selection for HMM and BLSTM based handwriting recognition of whiteboard notes,” International Journal of Pattern Recognition and Artificial Intelligence
8. K-F. Chain and D-Y Yeung, Recognizing on-line handwritten alphanumeric characters through flexible structural matching Pattern Recognition,
9. L. Rabiner, “A tutorial on hidden Markov models and selected applications in speech recognition,” In Proc. IEEE, vol. 77, no. 2
10. Tapan Kumar Bhowmik, Jean-Paul van Oosten, Lambert Schomaker, “Segmental K-Means Learning with Mixture Distribution for HMM Based Handwriting Recognition”
11. S. Smithies, K. Novins, and J. Arvo, “A handwriting-based equation editor,” Proc. Graphics Interface
12. Bunke, H.: Recognition of cursive roman handwriting–past, present and future. In: ICDAR ’03: Proceedings International Conference on Document Analysis and Recognition
13. http://www.cs.uwaterloo.ca/scg/mathbrush/mathdata.shtml
14. S. MacLean, The MathBrush character recognizer, Internal report for Symbolic Computation Group
15. Separating Handwritten Material from Machine Printed Text Using Hidden Markov Models; Jinhong K. Guo and Matthew Y. Ma
16. L. R. Rabiner, B.-H. Juang, S. E. Levinson, and M. M. Sondhi, “Recognition of isolated digits using hidden markov models with continuous mixture densities,” AT&T technical journal
17. Text Detection and Character Recognition in Scene Images with Unsupervised Feature Learning; Adam Coates, Blake Carpenter, Carl Case, Sanjeev Satheesh, Bipin Suresh, Tao Wang, David J. Wu, Andrew Y. Ng
18. A method for text localization and recognition in real-world images; Lukas Neumann and Jiri Matas

19. Richard Schwartz, Christopher LaPre, John Makhoul, Christopher Raphael, Ying Zhao "Language-independent OCR using a continuous speech recognition system"
20. H. El Abed ; V. Margner, "Comparison of Different Preprocessing and Feature Extraction Methods for Offline Recognition of Handwritten ArabicWords", Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)
21. Pattern Recognition and Machine Learning; Christopher M. Bishop
22. Globally Trained Handwritten Word Recognizer using Spatial Representation, Convolutional Neural Networks and Hidden Markov Models; Yosua Bengio, Yann Le Cun, Donnie Henderson
23. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition
24. Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber, ” Deep, Big, Simple Neural Nets for Handwritten Digit Recognition”, Neural Computation, December 2010
25. Alex Graves, Jurgen Schmidhuber, "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks"
26. “nolearn”, python package tutorial
27. D. C. Ciresan , Manno-Lugano, U. Meier , L. M. Gambardella , J. Schmidhuber, "Convolutional Neural Network Committees for Handwritten Character Classification", 2011 International Conference on Document Analysis and Recognition
28. A. Kaltenmeier , T. Caesar ; J. M. Gloger ; E. Mandler, "Sophisticated topology of hidden Markov models for cursive script recognition", Document Analysis and Recognition, 1993
29. Thomas Plötz , Gernot A. Fink. "Markov models for offline handwriting recognition: a survey", IJDAR 2009
30. Y. Nakayama. “A prototype pen-input mathematical formula editor”. In Proceedings of ED-MEDIA 93 – World Conference on Educational Multimedia and Hypermedia, pages 400{407, Orlando
31. M. Okamoto and A. Miyazawa. “An experimental implementation of a document recognition system for papers containing mathematical expressions”. In H. S. Baird, H. Bunke, and K. Yamamoto, editors, Structured Document Image Analysis
32. E. G. Miller and P. A. Viola. Ambiguity and constraint in mathematical expression recognition. In Proceedings of the Fifteenth National Conference on Articial Intelligence
33. A. Kacem, A. Bel□aid, and M. B. Ahmed. EXTRAFOR: Automatic EXTRAction of mathematical FORmulas. In ICDAR'99
34. H.-J. Lee and J.-S. Wang. Design of a mathematical expression recognition system. Pattern Recognition Letters, 18:289{298, 1997.
35. R. Marzinkewitsch. Operating computer algebra systems by handprinted input. In Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation
36. R. J. Fateman and T. Tokuyasu. Progress in recognizing typeset mathematics. In Proceedings of the SPIE, volume 2660
37. Y. A. Dimitriadis and J. L. Coronado. Towards an ART based mathematical editor, that uses on-line handwritten symbol recognition. Pattern Recognition, 28(6):1995