Lappeenranta University of Technology

BM40A0701 - Pattern Recognition

Practical assignment: Digits 3-D

10-12-2018

Authors

0539503 Pauline SCHMITT
0539590 Andrei TIURIN
(Group 14)

LUT University

Table of content

Introduction

This project was realized in the scope of the course 'Pattern Recognition'. The aim was to design a classifier that recognizes digits from 0 to 9, drawn in three dimensions with a finger and captured by a LeapMotion sensor.
The solution was implemented using Matlab, and is based on the knowledge developed during the course and some external references, listed at the end of this report and cited when necessary.
Our approach is a two layers neural network (NN), trained on 100 labelled samples for each digit. This required a data preprocessing and a feature extraction steps. After designing and training our classifier, we tested it to check it was working, and evaluate its performance. We then improved it by changing the values of the different parameters of the neural network.
This report describes our classification approach, explains our choices for the data preprocessing steps and the neural network parameters, and presents our results.

1. Data preprocessing

The input data for one digit is the 3-D location time series of the finger drawing the said digit. Several characteristics of this data make preprocessing necessary:

- **the distance between points**: all samples were not drawn at the same speed, and consequently the distance between sequential points is not constant. Some points can be very close to each other, almost forming a continuous line, while others will have a large space between each other. This varies both between different samples, and between points in a single sample.
- **the number of points**: this is also due to the drawing speed; the faster the digit was drawn, the less points the data sample will contain.
- **the range of the coordinates values**: depending on how big the digit was drawn and its position in the 3D reference space, the coordinates values will vary significantly for different samples collected of the same digit.
- **the inaccurate points**: on some samples, we can find points or group of points that are clearly not part of the digit. This is of course obvious to a human, but not to a computer.
- **the 3D representation of a 2D digit**: to collect data, humans were asked to draw digits in the air with their finger. We usually write these digits in two dimensions, so drawing them in the 3D space introduces a new variable in their representation.

To deal with these issues and minimize their impact on the classification, we performed several preprocessing steps:

1) **projection in two dimensions**: since all training samples were drawn in the same plane, we simply deleted coordinate z from the data, in order to have 2D coordinates.
2) **min-max normalization**: to deal with the problem of range, we normalized coordinates x and y respectively, using a min-max normalization (see formula below).

$x_{normalized} = \frac{x - min(x)}{max(x) - min(x)}$ where x (respectively y) is the vector of x coordinates (respectively y coordinates) of one sample.

3) **filling the gaps between points**: to deal with both problems of number of points and distance between points, we filled in the space between two sequential points, by aligned points, separated by a given distance. In the final solution, we add a point if the euclidean distance between two points is more than 0.01 .

2. Features extraction

Our solution doesn't only take into account the shape of the digit itself, but also the movement of the finger when drawing the digit, meaning the global direction towards which the finger is going.

Each 2D time series is converted into an image of size 15x15, i.e. a grid of which each cell contains the global direction of several data points. This direction is represented by a pair of values (x, y), that can be either -1, 0, or 1 (see figures 1). They are obtained from the data coordinates by summing all the differences between two sequential points, and then taking the sign of this sum:

$$cell\ value = sign(\sum_{n}(next\ point - current\ point)$$

where n is the total number of points.

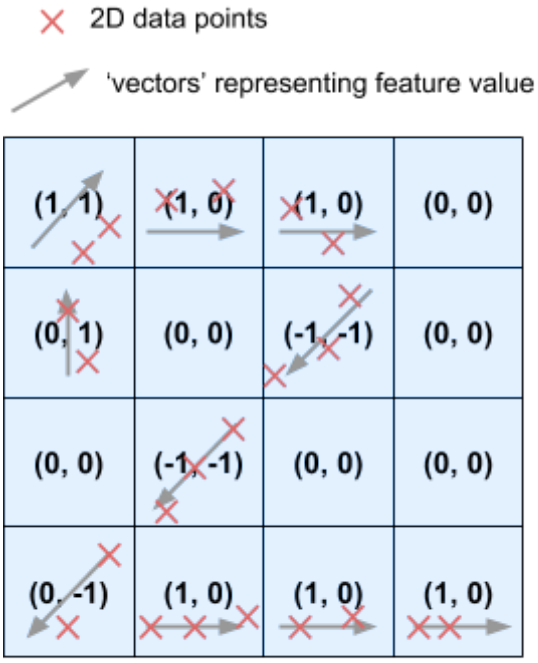

Figure 1: simplified representation of the image extracted from the 2D coordinates of one sample, for digit 2

This pair of values can be seen as a vector for better understanding of the principle, but in fact, what is generated is a 3D matrix with two 2D layers, each being a matrix of values for coordinates x and y, respectively.
A third layer in the matrix gives additional information about the order of points: we basically divide the data points in two groups, the first half that was drawn has a value of -1, and the last half that was drawn has a value of +1.

$$cell\ value = \begin{matrix} -1 & if \quad point\ index < \frac{total\ number\ of\ points}{2} \\ +1 & otherwise \end{matrix}$$

We end up with a 3D matrix, representing the image of the digit in 2D, with three layers: 1 for movement along coordinate x, one for movement along coordinate y, and one for the order of points.
This matrix is then put into a line vector to be given as an input to the neural network.

3. Classification

We used a neural network with one hidden layer of 10 neurons, and one output layer. The input of the NN is the line vector generated after the feature extraction step, and the output is a size 10 line vector of booleans, with 1 in the cell which index is *digit + 1* (because of Matlab convention of indexes starting at 1), and 0 in all other cells. For example, the output if the digit was classified as 5 would be [0 0 0 0 0 1 0 0 0 0] (see figure 2).
We used the *tanh* activation function for the hidden layer, and the *softmax* activation function for the output layer **[1]**.

$$softmax\,(x)_j = \frac{exp(x_j)}{\sum_{k=1}^{K} exp(x_k)}$$ where K is the dimension of vector x, and j = 1, … , K

During training, we give the NN the expected output vector, and the data samples for the corresponding digit. We compute the weights associated to all connections between neurons, and save them to an external file, given with our source code. These weights are then used during classification, when an unknown sample is given to the NN.

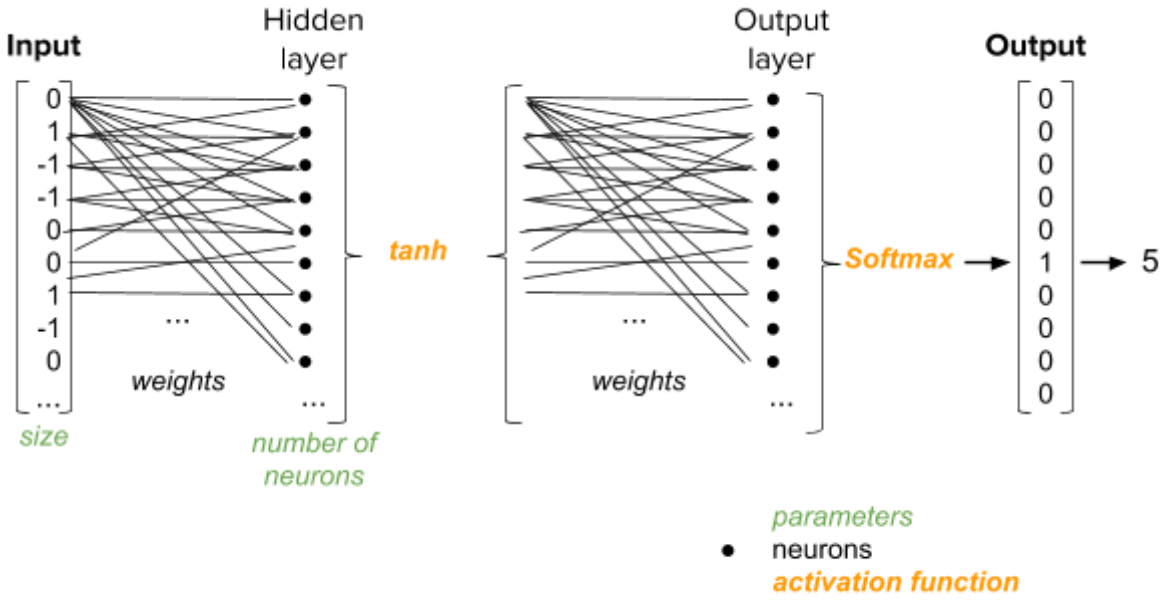

Figure 2: overview of the neural network solution

Several parameters can be modified in our solution:

- the space between two added points in the 3rd step of data preprocessing
- the size of the feature grid extracted from the data, i.e. the size of the NN input
- the number of neurons of the hidden layer

We tried different combinations of these values, in order to find the best one.

4. Results

To perform validation tests, we separated the dataset into two parts; we used the first one (70 samples per digit) for training, and the second one (30 samples per digit) for validation.

We simply compute the true detection rate (TDR) for each digit (see figure 3), and then the mean true detection rate for all digits. We reach a true detection rate of 93% after 10000 epochs of training with a learning rate of 0.001.

For one digit: $$TDR = \frac{number\ of\ recognized\ digits}{total\ number\ of\ validation\ samples}$$

For all digits: $$TDR_{total} = \frac{sum(TDR)}{10}$$

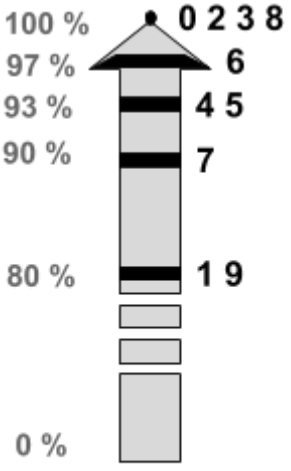

Figure 3: True detection rate, ordered by digit

Conclusion

In this project, we implemented a classifier for 3-D digits.
The 3D time series of each digit are preprocessed by projecting them in two dimensions, normalizing them, and adding points to have a more even distribution and fewer spaces between each point. Then we extracted the features, taking into account the direction and the order of the drawing. Finally, using a two layers neural network trained with labelled data, our algorithm is able to recognize digits with a global accuracy of 93%.
We could still improve this solution by several means; for example, we did not treat the problem of inaccurate data points in the preprocessing step. We could find a way to get rid of them, in order to have cleaner training samples. We could also perform more tests to find the best combination of parameter values.

References

[1] Machine Learning Cheatsheet, "Activation functions", available at *ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html* (last visited 09/12/2018)