

# **Assignment 2**

## **OCR – Optical Character Recognition**

### **Aprendizagem Computacional (MEI) CNSD (MIEB)**

Departamento de Engenharia Informática  
Faculdade de Ciências e Tecnologia  
Universidade de Coimbra

September 2018

This work covers the materials in Chapters 3, 4 and 5. Before starting, study these Chapters.

(adapted from a previous work plan authored by Jorge Henriques, [jh@dei.uc.pt](mailto:jh@dei.uc.pt))

## Index

1. **Digits recognition**
  - 1.1 Problem definition
  - 1.2 Characters definition
2. **Neural network architecture**
  - 2.1 Associative memory + classification
    - Associative memory
    - Classifier
  - 2.2 Classification
    - Classifier
3. **Matlab Implementation notes**
4. **Conclusions**

Before starting, read attentively all of this document and study the materials of Chapters 3, 4, 5.

Matlab documentation useful:

getting started nnet\_gs2018b.pdf

user's manual : nnet\_ug2018b (Chapters 4, 5, 13) (1,2,11 nnet\_ug2017b)

In the 2018b release the Neural Networks Toolbox has been renamed Deep Learning Toolbox. The 2018b first three chapters are about Deep Learning.

However to understand them the other chapters are needed. In this assignment the useful chapters are 4, 5, and 13, corresponding to the 1,2 and 11 of 2017b.

The getting started guide (nnet\_gs.pdf) is unchanged (only the name of the toolbox).

# 1. Digits recognition

## 1.1 Problem definition

Neuronal networks models will be developed for character recognition problems. The characters to be recognized are the 10 Arabic numerals:

**{1, 2, 3, 4, 5, 6, 7, 8, 9, 0}**

## 1.2 Characters definition

It is assumed that each character is defined by a matrix composed of binary (0/1) elements. The digits are defined as a 16x16 matrix. For example, the following matrix can represent the digit **0** supposedly manually traced by some user in some device.

```

0000011110000000
0001100011110000
00110000000011000
01010000000001100
01100000000000110
00100000000000010
00010000000000011
00011000000000001
00001000000000001
00001100000000001
00001100000000001
00000110000000001
00000011000000001
00000001100000010
0000000011001110
0000000000111000

```

We will use the supplied Matlab function **mpaper.m** to convert a character designed with the mouse into such a binary matrix. See inside it

the user's instructions. Then each 16x16 matrix of bits is converted to a column vector of  $16 \times 16 = 256$  elements. This is made by the *reshape* Matlab function that concatenates the columns of the matrix one after another.

## 2. Neural network architectures

Two neural networks architectures will be comparatively studied:

- i) both associative memory+classifier and
- ii) only a classifier.

### 2.1 Associative memory + classifier

Two neural networks are considered, serially connected as in Fig. 1:

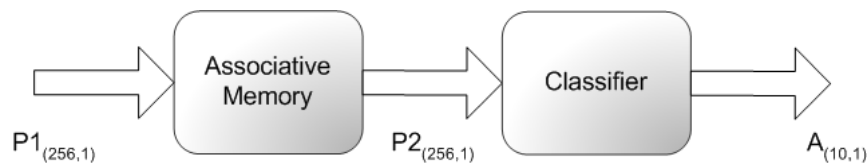


Fig. 1 Associative memory followed by a classifier

#### Associative memory

The first module, the associative memory, has as input the vector  $P1$  (dimension 256,1), that defines the character to be classified, corresponding to the binary matrix (16,16).

This associative memory can be seen as a "filter" or "corrector": if the input character is not perfect, the associative memory has the capacity to provide an output  $P2$  (dimension 256,1) that is potentially a "more perfect character".

For this give to the associative memory a target  $T$  of perfect characters using the supplied Matlab file PerfectArial.mat.

The associative memory is a neural network (see chapter 4 of the syllabus) consisting of:

- One single layer,
- Linear activation functions
- Without bias

Its output is computed by:

$$P_2 = W_p \times P_1$$

The weights,  $W_p$  (256,256), are evaluated using the pseudo-inverse method,

$$W_p = T \times \text{pinv}(P)$$

where  $T$  (256,  $Q$ ) are the desired  $Q$  outputs, for a given  $P$  inputs (256, $Q$ ).  $Q$  should be high, for example 500. If the size of  $P$  is too big for using *pinv*, then a different training function must be used.

### **Classifier**

The second neural network module is the classifier. The input ( $P_2$ ) is the output of the associative memory (AM). After training the AM, its output matrix is the input matrix of the classifier. The output ( $A$ ) is the class where the digit belongs.

For the digits to be classified { '1' '2' '3' '4' '5' '6' '7' '8' '9' '0'}, the following classes are assumed { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}. So the classifier has 10 outputs.

It is assumed that the classifier is a neural network consisting of:

- One single layer with 10 neurons (one for each class)
- A linear or non-linear activation function, namely
  - i) hardlim,
  - ii) linear,
  - iii) sigmoidal.
- With bias in each neuron.

It is characterized by:

$$A = f(W_N \times P_2 + b)$$

where matrix  $W_N$  has dimensions (10,256) and  $b$  dimensions (10,1). The input ( $P_2$ ) is a vector of dimension (256,1). The output ( $A$ ) has dimension (10,1).

Concerning the activation function there are three alternatives:

Harlim

$$A = \text{hardim}(W_N \times P_2 + b)$$

Linear

$$A = W_N \times P_2 + b$$

Sigmoidal

$$A = \text{logsig}(W_N \times P_2 + b)$$

For example, considering as input the digit defined in section 1.1 (zero) the output should belong to class 10 :

$$A = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The neural network parameters (weights and bias) should be evaluated using the perceptron rule, if harlim is used, or the gradient method if purlin or logsig (see chapter 4) are used.

Note that if `purlin` or `logsig` are used, the outputs of the classifier are not 0 or 1 but can be any real number in  $[-\infty, +\infty]$  or  $[0,1]$  respectively. This requires pos-processing in order to obtain the classification with 0-1. Several heuristics may be used, for example making 1 the higher and all the others zero, writing the appropriate Matlab script.

## 2.2 Classification

In this case only a neural network is considered, as shown in Fig. 2.

### Classifier

The characters (vector 256, 1) are directly provided to the classification system.



Figure 2. A single classifier

There is no pre-filtering (by any AM) of the data. The classifier must have a considerable generalization capability.

Recommendation: use the `train` or `adapt` function of NN toolbox only with train and validation sets. The validation sets prevent overfitting. The test set, to be created by the user using the `mpaper` facility, must be used with the network simulation function. This allows a better understanding of what is going on.

### 3. Matlab Implementation notes

First Read the Neural Networks Getting Started and Chapter 1 of the User's Guide.

One neural network is defined in Matlab, as an object, by a structure with fields to give all the needed data for training:

Architecture:

- network properties define the architecture (number of inputs, number of layers, bias connect, input connect, layer connect, output connect, target connect),
- sub-object properties define the details of the architecture (inputs, layers, outputs, targets, biases, input weights, layer weights, etc.),
- training properties (training function, training parameters). For each implemented training algorithm (and they are numerous) there are proper fields to give its needed parameters. For more see the Neural Networks toolbox users' guide (nnet-ug.pdf).

Numerical data is saved after Matlab 7 as .mat files. Save and load functions create and load .mat files by default. However there is a tool that can convert several other data types, namely .dat and ascii files. (see Neural Networks Toolbox Manual, Chapter 11 (2016a or after)- Network Object Reference).

In this work some auxiliary functions may be used. Read the distributed files and pay attention to the several comments and instructions in them.

**mpaper**: to write using the mouse a set of digits and to convert them to a binary vector of inputs (256, Q). Use this function only to create the input matrix, not to classify.

**grafica (X,Y,Z)**: to show until three draft digits **X, Y, Z** (256, 1)

**showim(P)**: to show all Q digits - **P**: (256, Q) or **T** (256, Q) in a 5x10 image matrix.

**ocr\_fun (data)**: calls the classifier and plots the numerals of the result in a 5X10 grid. Note that you must write the script of your classifier.

The file PerfectArial.mat contains the perfect coding of the ten digits in a structure called *Perfect*. To see them type showim(Perfect).

(**mpaper**, **showim** and **ocr\_fun** have been adapted from the Statistical Pattern Recognition Toolbox, (C) 1999-2003, written by Vojtech Franc and Vaclav Hlavac, <http://www.feld.cvut.cz>, Faculty of Electrical Engineering, Czech Technical University Prague )

**myclassify**: to perform the classification, and must be written by the user. Note that myclassify can be called from mpaper with the middle mouse button (or the shift+left button) with two purposes: to create the **P** matrices, and to classify the drawn characters.

### 3.1 Training Process

The user must define a training set with a sufficiently high number of cases, more that 500, to allow an effective training. The function mpaper allows to define 50 inputs at once. For more than 50, mpaper must be used several times to produce 50 each time, and then concatenate the several matrices of 50 columns to obtain the total matrix **P**.

The user must also define the target data set, with the desired output for each of the inputs. For the AM, the target is defined using the PerfectArial.mat file(note that the structure inside is named Perfect; if you want to see them try showim(Perfect), or grafica (Perfect(:,1), Perfect(:,2), Perfect(:,3)).

In the case of direct classification (without pre-filtering), then the **T** matrix is a set of ten 0/1 digits corresponding to the good answer for each of the inputs (in each **t** vector only one digit is 1, the others are 0).

To show the digits two Matlab functions are available (the first one gives more clear results):

**grafica(P(:,k))**, will show the k digit of the P matrix.

**showim(P)**, shows all digits in **P** (this one from the referred Statistical Pattern Recognition Toolbox). This file may show different levels of grey, with the scale between 0 (black) and 1 (white).

## **Classifier definition and training**

### **Neural network definition**

See the NN Starting Guide. Note that we have here a classification problem (pattern recognition problem). However the architectures in the starting guide are not adequate for our problem. You must build them with the appropriate functions. There are several possibilities (see the Technicalnote1.pdf in the website). Using `net=networks` (custom layer) gives the user total freedom to configure the network; so it is the recommended way.

### **activation function**

`hardlim`- binary

`purelin` – linear

`logsig` – sigmoidal

### **Neural network training**

There are two learning styles: the incremental and the batch.

#### **a) Incremental learning**

One input at a time is presented to the network, and the weights and bias are updated after each input is presented. There are several ways to do it:

`net=traininc(net,P,T)` : "trainc trains a network with weight and bias learning rules with incremental updates after each presentation of an input. Inputs are presented in cyclic order."

`net=trainr(net,P,T)` "trainr trains a network with weight and bias learning rules with incremental updates after each presentation of an input. Inputs are presented in random order."

When these learning methods are used, the algorithms must be iterative and they are implemented in the toolbox with names started by *learn* as for example:

- learnr – gradient rule
- learnr – gradient rule improved with momentum (see help)
- learnh – hebb rule (historical)
- learnhd- hebb rule with decaying weight (see help)
- learnwh- Widrow-Hoff learning rule

The learning function is specified by

`net.inputweights{1,1}.learnFcn='learnr'`, for example.

Incremental learning can also be done by

`net=adapt(net,P,T)`, but it is mandatory in this case that P and T be cell arrays (not matrices, as in the previous methods).

#### b) Batch training

`net=trainb(net,X,P)` "trainb trains a network with weight and bias learning rules with batch updates. The weights and biases are updated at the end of an entire pass through the input data."

`net=train(net,X,P)` , train by default is in batch mode.

In these methods the algorithms are in batch implementation, and their names start by train, as for example

trainr	gradient descent
trainrda	gradient descent with adaptive learning rate
trainrdm	gradient with moment

trainlm	Levenberg- Marquardt
trainscg	scaled conjugate gradient

Note that `learngd` and `traingd` both implement the gradient descent technique, but in different ways. The same for similar names. However `trainlm` has no incremental implementation, only batch.

In general batch learning is preferable to incremental learning, except for the learning rule, that is implemented only in incremental form with *learnp* and *learnpn*).

### Initialization

Initially the network parameters can be set by: (by default they are set to zero in the toolbox).

```
» W=rand(10,256); generates matrix 10x256 random in (0 1)
» b=rand(10,1);
» net.IW{1,1}=W;
» net.b{1,1}= b;
```

### Training parameters

It is possible to specify several parameters related to the training of the neural network. Some of them are: (depends on the NN Toolbox version).

```
» net.performParam.lr = 0.5; % learning rate
» net.trainParam.epochs = 1000; % maximum epochs
» net.trainParam.show = 35; % show
» net.trainParam.goal = 1e-6; % goal=objective
» net.performFcn = 'sse'; % criterion
```

### Training

To evaluate the optimal network parameters Matlab provides the **train** function.

```
net = train(net,P,T);
```

where  $P$  is the  $(256, Q)$  inputs and  $T$  the desired outputs  $(10, Q)$ .

The `train` function by default divides the data matrix  $P$  into three sets: training (70%), validation (15%), test (15%). To change these numbers use the `divide`

`[trainInd, valInd, testInd] = divideind(Q, trainInd, valInd, testInd)` separates targets into three sets: training, validation, and testing, according to indices provided. It actually returns the same indices it receives as arguments; its purpose is to allow the indices to be used for training, validation, and testing for a network to be set manually. Example:

```
[trainInd, valInd, testInd] = divideind(3000, 1:2000, 2001:2500, 2501:3000);
```

It is recommended that only training and validation sets be used for training (for example 85% training 15% validation).

After the training phase, the final weights and bias can be accessed by

```
» W = net.IW{1,1};
» b = net.b{1,1};
```

### Validation

To test the neural network it is available the **sim** function.  $P_t$  is the testing set created by the user with the support of `mpaper` (with at least 50 digits).

```
a = sim(net, Pt)
```

The test set must be different from the training set. It allows to measure the generalization capabilities of the network. If in the working group the two members have designed cases for the training stage, then also both must design cases for the testing phase.

## 4. Conclusions

The report should focus the following:

- **1. Data set**

- How does the data set influence the performance of the classification system?

- **2. Neural network architecture**

- Which architecture provides better results: only the classifier or the associative memory+classifier ?
- Which is the best activation function: hardlim, linear or logsig?

- **3. Results**

- Is the classification system able to achieve the main objectives (classification of digits)? Which is the percentage of well classified digits )
- How is the generalization capacity? Is the classification system robust enough (to give correct outputs when new inputs are not perfect)? Which is the percentage of well classified new inputs ?

Enjoy yourself with this interesting work. Many devices of daily use recognize characters (hand-written or not) based on a neural network classifier.

## **Report format:**

The delivered material (as the report of this work) must allow the reader to design in mpaper a set of digits and then to test the delivered trained classifiers in an easy way, with the support of a GUI (the user draws in mpaper a set of characters, then with the right mouse classifies them and the result is presented to him in a 5x10 grid, as in mpaper, or creates a test file with mpaper, names it, and then calls it from the GUI and tests the

chosen classifier in a field of the GUI). At least the two best classifiers must be called from the GUI. The software must include the grafication of the results in a grid similar to the one of mpaper: then two grids will be presented to the reader, the input one and the results in a way that the reader can easily see which ones are well classified and which ones are not.

The delivered report file must be named **AC2018OCRPLxGy.pdf**, where x is the number of the class and y is the number of the group in the class, to ease its identification.

All the files, including the pdf report and data files, must be packed in a .zip or .rar that by uncompressing will create the directory structure adequate for running the scripts. The compressed file (zip or rar) must have the name **AC2018OCRPLxGy.zip[rar]**

Please make easy the tasks of the reader and the user of your software (many thanks for that).

**Remark: for GUI see** App Building documentation (GUIDE, >guide, or App Designer-> appdesigner).

---