1 2 9 0

UNIVERSIDADE D COIMBRA

MEB, MEI, MES, MIEBIOM

Aprendizagem Computacional/Machine Learning

Computação Neuronal e Sistemas Difusos/ Neural Computation and Fuzzy Systems

Departamento de Engenharia Informática

2020/2021

ASSIGNMENT 2A

OCR – Optical Character Recognition

1. Context and document structure

The main goal of this work is to develop a neural network based system for Optical Character Recognition (OCR), specifically for digits recognition. This work should be developed in MATLAB (including the Image Processing Toolbox) and covers the materials in Chapter 4. It is adapted from a previous work plan by **Jorge Henriques** and **António Dourado** (jh@dei.uc.pt; dourado@dei.uc.pt). Before starting you should:

- study Chapter 4,
- read this document until the end,
- read the introduction in the MATLAB files, namely MATLAB documentation useful (release 2020b):
    - getting started nnet_gs.pdf (pages 1-40 and following)
    - user's guide: nnet_ug (Chapters 19, 20, 27)

The assignment is split into two sub-assignments: 2A) first 2 weeks, detailed in this document, and 2B) second 2 weeks, detailed in a latter to be available document. Students should implement their solutions and answer the challenges in a **report** that should be submitted on Inforestudante together with the **MATLAB code files** before the deadline: 17/11/2021 (different submission links for 2A and 2B will be available).

<u>Conditions</u>: Groups: two elements of the same PL class

<u>Duration</u>: 4 weeks (plus 2 weeks with no classes)

<u>Workload</u>: 20h per student

Document structure:

2. Digits recognition

2.1 Problem definition

The problem to be addressed in this assignment is the development of Neural network models for handwritten character recognition. The characters to be recognized are the 10 Arabic numerals:

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

2.2 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.

0	0	0	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0
0	0	0	**1**	**1**	0	0	0	**1**	**1**	**1**	**1**	0	0	0	0
0	0	**1**	**1**	**0**	0	0	0	0	0	0	**1**	**1**	0	0	0
0	**1**	0	**1**	0	0	0	0	0	0	0	0	**1**	**1**	0	0
0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	**1**	**1**	0
0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	**1**	0
0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	1	**1**
0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	0	**1**
0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	**1**
0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	**1**
0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	**1**
0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	0	**1**
0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	0	0	**1**
0	0	0	0	0	0	0	**1**	**1**	0	0	0	0	0	**1**	0
0	0	0	0	0	0	0	0	**1**	**1**	0	0	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	0	0	0

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 16x16=256 elements. This is made by the ***reshape*** MATLAB function that concatenates the columns of the matrix one after another.

3. Neural Network Architectures

Two neural networks architectures will be comparatively studied:

i. both filter and classifier

ii. only classifier.

3.1 Filtering and Classification

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

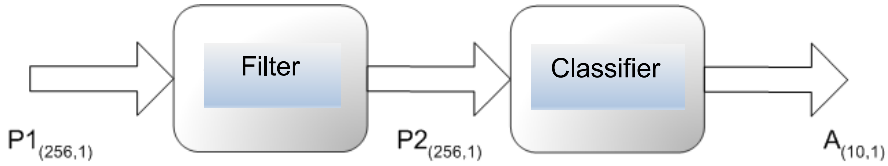

Figure 1 A filter followed by a classifier

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

a) Associative memory as filter

The 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, *Wp* (256,256), are evaluated using the pseudo-inverse method, $W_P = T \times pinv(P)$, 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.

b) Binary perceptron as filter

Instead of the associative memory you can use a binary perceptron with 256 **hardlim** neurons. For this give to the perceptron the same target T of perfect characters using the supplied MATLAB file **PerfectArial.mat**. For training use the function net.trainFcn='*trainc*' and net.adapFcn='*learnp*'.

Classifier with one layer

The second neural network module is the classifier with a single layer. The input (P2) is the output of the associative memory (AM) or the binary perceptron. After training, 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}. Hence, 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
hardlim,
linear,
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:

Hardlim: $\boldsymbol{A} = \boldsymbol{hardlim}(\boldsymbol{W_N} \times \boldsymbol{P_2} + \boldsymbol{b})$

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 2.2 (zero) the output should belong to class 10 :

$$A = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$$

The neural network parameters (weights and bias) should be evaluated using the perceptron rule, if hardlim 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 [-inf, +inf] or [0,1] respectively. This requires post-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. You can also use a layer with softmax activation function (see help softmax).

3.2 Classification

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

Figure 2 A single classifier

Classifier with one or two layers

The characters (vector 256, 1) are directly provided to the classification system. There is no pre-filtering of the data. The classifier must have a considerable generalization capability.

a) Try classifiers with one layer (with 10 neurons)

b) Try a classifier with two layers: the hidden layer and the output layer. The output layer has 10 neurons. The hidden layer has a number of neurons that you must fix, in principle more than 10.

c) You can also try the patternet (see nnet_gs page 1-73 and following)

Compare the results of the two architectures.

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.

Remark on building the data matrices P and T:

It will be easier if you draw the characters in matrix P by blocs of 10 with the order (1,2,3,4,5,6,7,8,9,0) as is given in **PerfectArial**. This way the target mmatrix T for the filter will be the horizontal concatenation of PerfectArial as many times as needed. The target for the classifier will be the identity matrix of dimension 10 concatenated with itself as many times as needed.

4. Report

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:

- the filter + classifier?

- only the classifier with one layer?

- only the classifier with two layers?

- what is the advantage (if any) of the softmax layer?

- 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 simple 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. Figure 3 shows an example of a possible GUI.

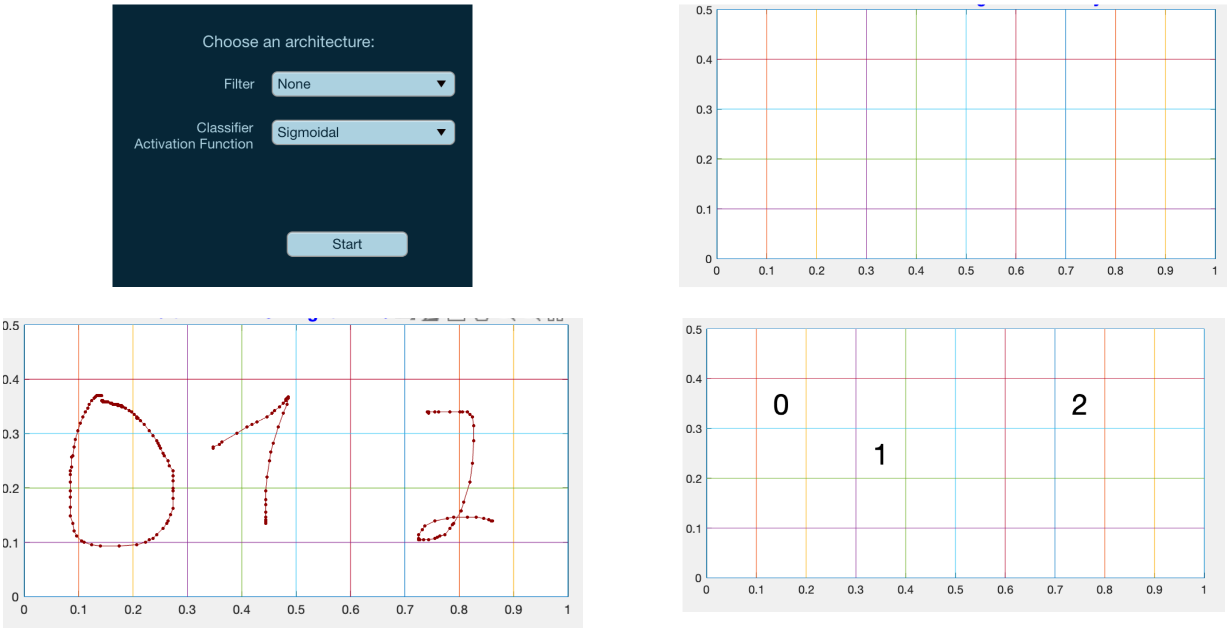

Figure 3 Example of a possible GUI

The delivered report file must be named **ML2020OCRPLxGy.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 **ML2020OCRPLxGy**.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).

5. APPENDIX - MATLAB Implementation Notes

First Read the Neural Networks Getting Started (page 40 and following, version 2020b) and Chapter of the User's Guide Chapter 19 and 20, version 2020b. 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 .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 Chapter 28 Neural Network Object Reference).

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

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.

5.1 Training Process

The user must define a training set with a sufficiently high number of cases, more than 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 filter, 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).

5.1.1 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 note pdf file). 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 training

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.trainFcn='trainc'
- ***net=train(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.trainFcn=' trainr'
- ***net=train(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 training methods are used, the learning algorithms must be iterative, and they are implemented in the toolbox with names started by *learn* as for example:
learnp – perceptron rule
learnpn – normalized perceptron rule
learngd – gradient rule
learngdm – 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 algorithm is specified by **net.adapFcn='learnp',** 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.trainFcn='trainb' "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,P,T) , train by default is in batch mode.

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

traingd	gradient descent
traingda	gradient descent with adaptive leaning rate
traingdm	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. The training functions are specified by net.trainFcn='trainlm' for example.

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

Initialization

Initially the network parameters can be initialized randomly.

For the perceptron the default initializations are zeros.

For the case of one layer with 10 outputs and 256 inputs:

```
» W=rand(10,256); generates matrix 10x256 random in (0 1)
» b=rand(10,1);
To initialize W and b randomly in the interval (a,b)
» W=a+(b-a).*rand(10,256)
» b=a + (b-a).*rand(10,1);
» net.IW{1,1}=W;
» net.b{1,1}= b;
For example in (-1, 1) to allow negative and positive weights and bias at
start:
» W=-1 +2.*rand(10,256);
» b=a + (b-a).*rand(10,1);
```

Training parameters

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

That 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 weighs and bias can be accessed by (for the case of one layer):

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

Validation

To test the neural network, i the **sim** function is available. Pt 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.