Lab assignment 1: Multilayer perceptron implementation

Academic year 2021/2022

Subject: Introduction to computational models
4th course Computer Science Engineering Degree (University of Córdoba)

22nd September 2021

Abstract

This lab assignment serves as familiarisation for the student with neural network computational models, in particular, with the multilayer perceptron. To do this, the student must implement the basic back-propagation algorithm for multilayer perceptron and check the effect of the different parameters (network architecture, moment factor, use of validation set, decrease of the learning rate for each layer and so on). Delivery will be made using the task in Moodle authorized for this purpose. All deliverables must be uploaded in a single compressed file indicated in this document. The deadline for the submission is **12th October**. In case two students submit copied assignments, neither of them will be scored.

1 Introduction

The work to be done in this lab assignment consists on implementing the back-propagation algorithm to train a multilayer perceptron for a specific problem.

To do this, a program capable of carrying out this training must be developed, with different options regarding its parametrisation. This program will be used to train models able to predict, with the highest accuracy, the objective variable(s) of a set of databases, available on Moodle, analysing the obtained results. **This analysis will greatly influence the qualification of this assignment.**

In the statement of the assignment, indicative values are provided for all parameters. However, it will be positively evaluated if the student finds other values for these parameters that achieve better results. The only conditions are the maximum number of iterations, 1000 for the outer loop and N for the inner loop, therefore, it can not overcome $1000 \cdot N$, where N is the number of patterns from the dataset and that the values for the remaining parameters should be constant for all the datasets or, in any case, they should depend on the aforementioned size.

Section 2 describes a series of general guidelines when implementing the back-propagation algorithm. Section 3 explains the experiments to be carried out once the algorithm is implemented. Finally, section 4 specifies the files to be delivered for this assignment.

2 Implementation of the back-propagation algorithm

Follow the instructions on the class slides, which provide a brief overview, pseudocode and the overall strategy to be followed for the implementation of the algorithm. Some characteristics that need to be clarified are the following:

- *Network architecture*: We intend to develop a generic algorithm, where the structure of the multilayer perception is free and to be chosen by the user. The number of layers H must be

$H \geq 2$, i.e. at least there must be an input layer (layer 0), a hidden layer (layer 1) and an output layer (layer 2), but the number of hidden layers may be higher. In addition, the user can specify the number of neurons in each of the hidden layers (the number of input and output neurons is determined by the problem considered). The same number of neurons will be taken for all the hidden layers.

- *Neurons type*: All the neurons, except those in the input layer, will be sigmoid and will have a bias. Therefore, its expression will be:

$$out_j^h = \frac{1}{1 + \exp(-w_{j0}^h - \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1})}.$$

- *Weights update*: we will use a learning rate of $\eta = 0.1$ and a momentum factor of $\mu = 0.9$. It must be tried a decreasing learning rate for each layer, with the following expression:

$$\eta_h = F^{-(H-h)}\eta, h \in \{1, \dots, H\},$$

  being F a decreasing factor established by the user (for instance, $F = 2$) and eta the original learning rate ($\eta = 0.1$).

- *Operation mode*: For this first lab assignment, the algorithm will work in online mode, i.e. for each training patter (inner loop), we will compute the error and modify the weights according to that error. Once all the training patterns have been processed, we will check the stop condition of the external loop and start again with the first pattern, only if the condition was not met.

- *Datasets*: The algorithm will work with a training and a testing set. The weight adjustment will be made using the training set and, in each iteration of the external loop, we will show the error made by the neural network when the training dataset is considered. The best way to know if the algorithm has been correctly implemented is to check if this training error converges and decreases each iteration. In addition, when the algorithm finishes, we will show the error performed by the network in the testing set. Optionally, a subset of the training set could be used as validation set. This set is used to stop the algorithm before it is overtraining (see *stopping condition* bellow). In any case, the test set can not be used nor to adjust weights nor to decide when to stop the algorithm.

- *Stopping condition*: Training will stop if any of these three conditions occur:

  - More than $1000 \cdot N$ weights adjustments have been made in the network, where N is the number of patterns of the dataset. This is, 1000 iterations for the outer loop, each of which involves N iterations of the inner loop (one iteration for each pattern).
  - If for 50 iteration in a row the training error does not decreased (or increased). A tolerance of 10^{-5} should be used to perform this check, i.e. if the training error decreases by an amount less than or equal to 10^{-5}, then, we consider that the error has not decreased.
  - If for 50 iteration in a row the validation error does not decreased (or increased). This stopping condition will only be taken into account if the user decides to use a validation set (known as early stopping). In this case, we will use $v \times 100\%$ of the training patters for validation (they will not be used in the training step), where $v \in [0, 1)$ is a value to be decided by the user. A tolerance of 10^{-5} should be used to perform this check, i.e. if the validation error decreases by an amount less than or equal to 10^{-5}, then, we consider that the error has not decreased. If $v = 0\%$, this stopping condition is not taken into account.

- *Weights copies*: Depending on the problem, the error surface can be very complex and sometimes the algorithm can jump to a point where it is unable to move and minimise the error. This is why we must keep a backup of the network weights that have led so far to the slightest error. So, before stopping the algorithm, we will always restore the backup.
- *Seeds for random numbers*: The algorithm we are running is an stochastic algorithm, i.e. the output of the neural network can depend on the initial value of the weights (first point considered on the error surface). To better analyse its behaviour, we are going to remove the bias introduced by the seed of the random numbers. Otherwise, the conclusions obtained may not be valid from a general point of view. One way to achieve this is carrying out several runs using different seeds and computing the average results over all the runs, in order to increase the fidelity of the behaviour. This is why the training stage will be repeated five times, using the seeds 1, 2, 3, 4 and 5, and, after this, we will show the training and testing errors for each run and the average and standard deviation of these five runs.

3 Experiments

We will test different configurations of the neural network and execute each configuration with five seeds (1, 2, 3, 4 and 5). Based on the results obtained, the average and standard deviation of the error will be obtained. The MSE error must be calculated for both the training set and the test set. The MSE is defined as follows:

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\left(\frac{1}{k}\sum_{o=1}^{k}(d_{po} - o_{po})^2\right), \tag{1}$$

where N is the number of pattern in the considered dataset (training or testing), k is the number of outputs, d_{po} is the target value for patter p and the output variable o and o_{po} is the predicted value.

To assess how the implemented algorithm works, we will use four datasets:

- *XOR problem*: this dataset represents the problem of non-linear classification of the XOR. The same file will be used for train and test.
- *Sine function*: this dataset is composed by 120 training patterns and 41 testing patterns. It has been obtained adding some random noise to the sine function (see Figure 1).
- *Quake dataset*: this dataset is composed by 1633 training patterns and 546 testing patterns. It corresponds to a database in which the objective is to find out the strength of an earthquake (measured on the Richter scale). As input variables, we use the depth of focus, the latitude at which it occurs and the longitude [1].
- *Parkinson dataset*: this dataset is composed by 4406 training patterns and 1469 testing patterns. It contains, as inputs or independent variables, a series of clinical data from patients with Parkinson's disease, including biometric measurement data from their voice. Furthermore, as output or dependent variables, it includes the motor value and the UPDRS (Unified Parkinson's Disease Rating Scale) [2].

In order to take advantage from the sine function characteristics, the input variables have been normalised between $[-1, 1]$. The output variable, has been normalised between $[0, 1]$.

A table for each dataset should be constructed, comparing the average and standard deviation of the train and test sets in terms of MSE (for the XOR problem, values for the train set will suffice) for the different configurations used. At least, the following configurations should be tested:

[1]see `https://sci2s.ugr.es/keel/dataset.php?cod=75` to seek more information.

[2]Check `http://archive.ics.uci.edu/ml/datasets/Parkinsons+Telemonitoring` to seek more information

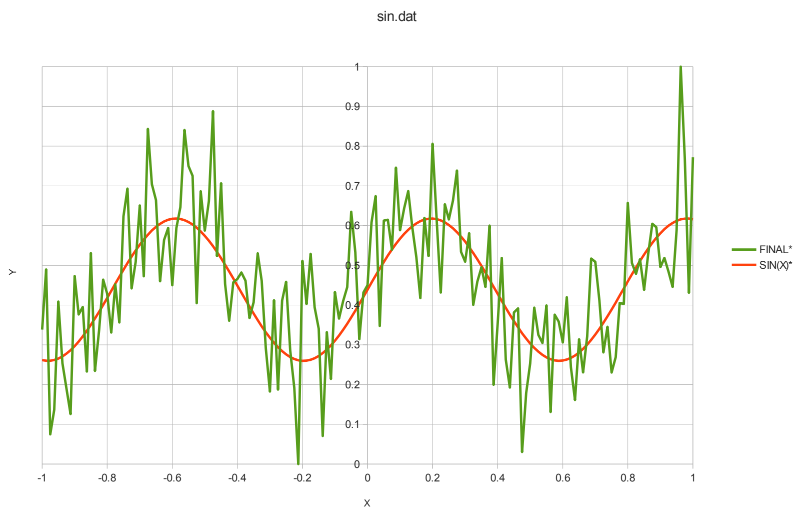

Figure 1: Representation of the data included for the sine function estimation problem.

- *Network architecture*: For this first step, we will use a momentum factor, we will not use a validation set ($v = 0.0$) and we will not use decreasing of the learning rate ($F = 1$). A total of 12 architectures should be tested:
  - With a hidden layer: $\{n : 2 : k\}$, $\{n : 4 : k\}$, $\{n : 8 : k\}$, $\{n : 32 : k\}$, $\{n : 64 : k\}$ y $\{n : 100 : k\}$.
  - With two hidden layers: $\{n : 2 : 2 : k\}$, $\{n : 4 : 4 : k\}$, $\{n : 8 : 8 : k\}$, $\{n : 32 : 32 : k\}$, $\{n : 64 : 64 : k\}$ y $\{n : 100 : 100 : k\}$.
- Once we have decided the best network architecture for each problem, we will try all the combinations for these two parameters: $v \in \{0.0; 0.15; 0.25\}$ y $F \in \{1; 2\}$.
- NOTE1: for the XOR dataset, $v = 0.0$ (no validation) will be considered.
- NOTE2: note if when we activate the validation ($v = 0.15$ o $v = 0.25$), the number of average iterations decreases with respect to not considering validation ($v = 0.0$). This implies a lower computational cost and, therefore, it is an advantage.

As a guideline, the training and generalisation error obtained by a linear regression (using Weka) is shown below:

- *XOR problem*: $MSE_{\text{train}} = MSE_{\text{test}} = 0.25$.
- *Sine function*: $MSE_{\text{train}} = 0.02968729$; $MSE_{\text{test}} = 0.03636649$.
- *Quake dataset*: $MSE_{\text{train}} = 0.03020644$; $MSE_{\text{test}} = 0.02732409$.
- *Parkinson dataset*: $MSE_{\text{train}} = 0.043390$; $MSE_{\text{test}} = 0.046354$.

The student should be able to improve this error values with some of the configurations.

3.1 File format

The files containing the datasets will follow the following format:

- In the first line, the total number of inputs (n), the total number of outputs (k) and the total number of patterns N is included.

- Then, each line is a different pattern, including $n + k$ real values. For the pattern/line p:
  - The first n values will be the pattern inputs, this is, $\mathbf{x}_p = \{x_{p1}, \dots, x_{pn}\}$.
  - The following k values will be the pattern outputs, this is, $\mathbf{d}_p = \{d_{p1}, \dots, d_{pk}\}$.

  An example of this type of file (for the XOR problem) is the following:

```
2 1 4
 1 -1 1
-1 -1 0
-1  1 1
 1  1 0
```

4 Assignments

The files to be submitted will be the following:

- Report in a `pdf` file describing the programme implemented, including results tables and their analysis.
- Executable file and source code.

4.1 Report

The report for this lab assignment must include, at least, the following content:

- Cover with the lab assignment number, its title, subject, degree, faculty department, university, academic year, name, DNI and email of the student
- Index of the content with page numbers.
- Description of the neural network models used (architecture and layer organisation) (**1 page maximum**).
- Pseudocode description of the back-propagation algorithm and all those relevant operations. The pseudocode must necessarily reflect the implementation and development done and not a generic description extracted from the slides or any other source. (**3 pages maximum**).
- Experiments and results discussion:
  - Brief description of the datasets used.
  - Brief description of the values of the parameters considered.
  - Results obtained, according to the format specified in the previous section.
  - Discussion/analysis of the results. The analysis must be aimed at justifying the results obtained instead of merely describing the tables. Take into account that this part is extremely decisive in the lab assignment qualification. The inclusion of the following comparison items will be appreciated:
    - ∗ Convergence charts: they reflect, on the x-axis, the iteration number of the algorithms, and, in the y-axis, the training error, the validation error and the test error.
    - ∗ Analysis of the neural network model obtained for the XOR problem, using the simplest architecture. Include the architecture specifying all the weights for the links. Check which is the value predicted by this model for the outputs and compare against the target value.

    * Any other graphic or analysis that the student deems appropriate (e.g. the predicted sine function performed by the best neural network using a format similar to Figure 1.).

- Bibliographic references or any other material consulted in order to carry out the lab assignment different to the one provided by the lecturers (if any).

Although the content is important, the presentation, including the style and structure of the document will also be valued. The presence of too many spelling mistakes can decrease the grade obtained.

4.2 Executable and source code

Together with the report, the executable file prepared to be run in the UCO's machines (concretely, test using `ssh` on `ts.uco.es`) must be included. In addition, all the source code must be included. The executable should have the following characteristics:

- Its name will be `la1`.
- The programme to be developed receive nine arguments on command line (that could appear in any order)[3]:
  - Argument `t`: Indicates the name of the file that contains the training data to be used. This argument is compulsory, and without it, the program can not work.
  - Argument `T`: Indicates the name of the file that contains the testing data to be used. If it is not specified, training data will be used as testing data.
  - Argument `i`: Indicates the number of iterations for the outer loop. If it is not specified, use 1000 iterations.
  - Argument `l`: Indicates the number of hidden layers of the neural network. If it is not specified, use 1 hidden layer.
  - Argument `h`: Indicates the number of neurons to be introduced in each hidden layer. If it is not specified, use 5 neurons.
  - Argument `e`: Indicates the value for the eta (η) parameter. By default, use $\eta = 0.1$.
  - Argument `m`: Indicates the value for the mu (μ) parameter. By default, use $\mu = 0.9$.
  - Argument `v`: Indicates the ratio of training patterns to be used as validation patterns. By default, use $v = 0.0$.
  - Argument `d`: Indicates the value for the decreasing factor (F in the slides) to be used for each of the layers. By default, use $F = 1$.
- Optionally, another argument could be included to save the configuration of the trained model (it would be necessary to obtain the predictions for the Kaggle competition):
  - Argument `w`: Indicates the name of the file in which the configuration will be stored and the value of the weights of the trained model.
- An example of execution can be seen in the following output:

```
i02gupep@NEWTS:~/imc/imc2021/workspace/la1/Debug$ ./la1 -t ../train_xor.dat -T ../
    test_xor.dat -i 1000 -l 1 -h 10 -e 0.1 -m 0.9 -v 0.0 -d 1.0
**********
SEED 1
**********
Iteration 1      Training error: 0.298534       Validation error: 0
Iteration 2      Training error: 0.287343       Validation error: 0
...
```

[3]Use the function `getopt()` from `libc` to process the input sequence.

```
Iteration 1000   Training error: 0.00964613      Validation error: 0
NETWORK WEIGHTS
===============
Layer 1
------
0.633863 -0.403033 0.773307
0.469708 0.669090 -0.478398
-0.344827 0.587652 -0.381660
-0.838194 0.833304 0.683623
-0.652504 0.552441 0.931250
1.037431 0.353168 0.532612
-0.936836 -0.204622 -0.874316
-2.507921 -2.810956 2.605680
-2.582238 -2.257553 -2.377693
-2.176769 2.170193 -2.377776
Layer 2
------
-0.291562 -0.309591 -0.436306 -1.468180 -0.887955 0.491371 -0.703266 4.389247
    -4.066886 2.783916 -0.831752
Desired output Vs Obtained output (test)
=========================================
1 -- 0.891937
0 -- 0.09749
1 -- 0.914304
0 -- 0.100294
We end!! => Final test error: 0.00964613
**********
SEED 2
**********
Iteration 1      Training error: 0.25546         Validation error: 0
Iteration 2      Training error: 0.254574        Validation error: 0
...
**********
SEED 5
**********
Iteration 1      Training error: 0.268381        Validation error: 0
...
Iteration 1000   Training error: 0.00813402      Validation error: 0
NETWORK WEIGHTS
===============
Layer 1
------
-2.621413 -2.489292 2.562251
-1.247278 -1.357582 -1.320136
-0.106533 -0.523225 -0.208347
1.382290 -1.410529 -1.396242
1.668992 -1.646763 -1.711076
-1.818481 -1.982124 -1.906150
-0.163134 -0.476374 -0.384511
0.097310 -0.311289 -0.365728
-1.168762 1.072397 1.266540
1.949929 -1.952045 1.882673
Layer 2
------
4.172039 -1.404449 -0.930334 1.370385 1.978051 -2.735954 -0.117170 -1.000121
    -0.972900 -2.964386 0.838221
Desired output Vs Obtained output (test)
=========================================
1 -- 0.915715
0 -- 0.0884255
1 -- 0.902992
0 -- 0.0905678
We end!! => Final test error: 0.00813402
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.00895706 +- 0.000701482
```

```
Test error (Mean +- SD):          0.00895706 +- 0.000701482

i02gupep@NEWTS:~/imc/imc2021/workspace/la1/Debug$ ./la1 -t ../train_parkinsons.dat
    -T ../test_parkinsons.dat -i 1000 -l 2 -h 32 -e 0.1 -m 0.9 -v 0.15 -d 1.0
**********
SEED 1
**********
Iteration 1        Training error: 0.0453154          Validation error: 0.0463736
Iteration 2        Training error: 0.0442438          Validation error: 0.0452932
...
We end!! => Final test error: 0.00972168
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.00775884 +- 0.00468669
Test error (Mean +- SD):          0.0111319 +- 0.00371563

i02gupep@NEWTS:~/imc/imc2021/workspace/la1/Debug$ ./la1 -t ../train_quake.dat -T
    ../test_quake.dat -i 1000 -l 1 -h 8 -e 0.1 -m 0.9 -v 0.15 -d 1.0
**********
SEED 1
**********
Iteration 1        Training error: 0.0302003          Validation error: 0.0312965
Iteration 2        Training error: 0.0301611          Validation error: 0.0312417
...
We end!! => Final test error: 0.0272729
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.0297427 +- 0.000510125
Test error (Mean +- SD):          0.0271742 +- 0.000119291
```

4.3 [OPTIONAL] Obtaining the predictions for Kaggle

The same executable of the assignment will allow obtaining the predictions for a given dataset. This output must be saved in a `.csv` file that must be uploaded to Kaggle to participate in the competition (check the file format of `sampleSubmission.csv` on Kaggle). This prediction mode uses different parameter than those mentioned previously:

- Argument `p`: Flag indicating that the program will run in prediction mode.
- Argument `T`: Indicates the name of the file containing the test data to be used (`test_kaggle.dat`).
- Argument `w`: Indicates the name of the file containing the configuration and the values for the weights of the trained model that will be used to predict the outputs.

Below is an example of how the training mode is executed using the parameter `w`, which saves the configuration of the model. In this case, we have used all the data for training and generalising, but it is highly recommended to divide the training set into two different subsets (training and validation).

```
i02gupep@NEWTS:~/imc/practica1/Debug$ ./la1 -t train.dat -T train.dat -w weights.txt -i
    100 -l 1 -h 32 -e 0.1 -m 0.9 -v 0.15 -d 1.0
**********
SEED 1
**********
Iteration 1        Training error: 0.015044           Validation error: 0.0122674
Iteration 2        Training error: 0.0135553          Validation error: 0.0109302
Iteration 3        Training error: 0.0118763          Validation error: 0.0094184
...
We end!! => Final test error: 0.00119128
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
```

```
Train error (Mean +- SD): 0.00140273 +- 0.000219986
Test error (Mean +- SD):          0.00144491 +- 0.000222772
```

Below is an example of the output using the prediction mode:

```
i02gupep@NEWTS:~/imc/practica1/Debug$ ./la1 -p -T test_kaggle.dat -w weights.txt
Id,Predicted
0,0.0220356
1,0.0820215
2,0.0241959
...
3497,0.103475
3498,0.00675393
3499,0.00506246
```