



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Study of efficient numerical tools for Machine Learning

Document:

ANNEX I: Code used and unified
modeling language

Author:

Antonio Darder Bennassar

Director /Co-director:

Àlex Ferrer Ferre

Degree:

Bachelor's degree in Aerospace
Vehicle Engineering

Examination session:

Spring

BACHELOR FINAL THESIS

Contents

List of Figures	I
1 Introduction	1
2 Code programming and UML	3
3 How to run the code	4
4 Conclusions	6

List of Figures

1	Structure of the repository	1
2	Decomposition of the principal folders	2
3	Unified Modeling Language (UML) of the code	3

1 Introduction

The aim of this document is to present the codes developed to build the artificial neural networks with which the results from the report were obtained. To do so, the strategy used to code the network along with unified modeling language (UML) diagrams are shown. All the code can be downloaded from this [github's link](#).

Object Oriented Programming has been one of the pillars of the project to produce a clean and extendable code. All the principles used have been taken from "*The Object-Oriented Thought Process*", Matt Weisfeld 2009.

Folder Structure

The repository contains 4 folders and one *README.txt* file, indeed each of these four folders has a file called *README.txt* explaining a little bit the contents of the folder. The 4 folders are the following

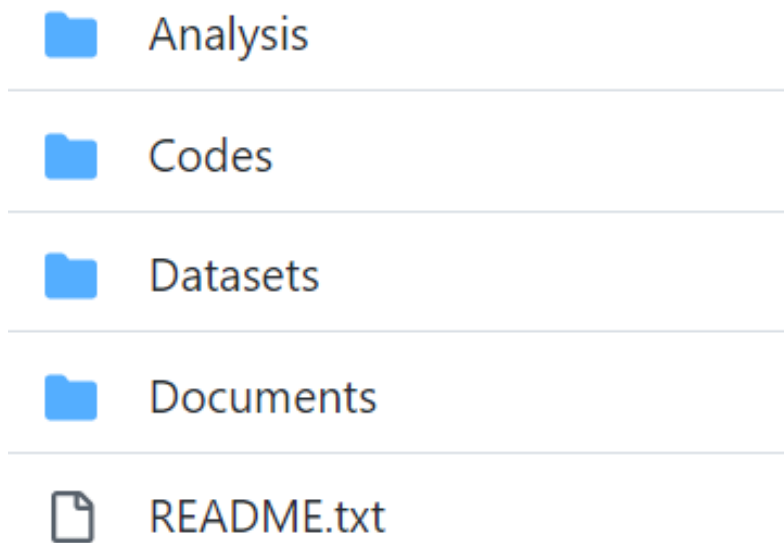


Figure 1 Structure of the repository

The contents of the 4 main folders are shown in figure 2, the first folder *Documents*, is provided for easy access, it contains the report of the project, the budget and this document.

The folder *Datasets* contains the 7 datasets used in the project. It is also provided the folder containing the images used to build the two datasets of "Flowers". One is the 32x32 rescaled images and the other is the dct with 3015 features truncation. All the files are in format csv which is the accepted by the class Data.

The folder *Codes* contains all the matlab functions/classes created to develop the thesis. It also contains the "datasets.mat" which is a vector of strings containing the name of the datasets for convenience.

Finally the folder *Analysis* contains a sample script "GeneralCall.m" which can be used to summon any dataset from the folder *Datasets* and train it with different neural networks and different optimization tools. It also contains "IrisTest.m" which is a test presented in the report of the thesis in section 7. It trains the Iris dataset with a fixed structure and optimization algorithm. Lastly it contains 6 scripts enumerated from A to F, these scripts are the ones that were run to obtain the different plots shown in the report section 8. They are different analysis on different parameters which affect the model performance.





































 A_ActFuntion.m	 flower_photos
 B_BatchSize.m	 4circles.csv
 C_OptimizationParams.m	 ConCircles.csv
 D_Regularization.m	 Flowers3232.csv
 E_MNIST.m	 FlowersDCT.csv
 F_Flowers.m	 Gender.csv
 GeneralCall.m	 Iris.csv
 IrisTest.m	 MNIST.csv
 README.txt	 Microchip.csv
	 README.txt
(a) Analysis	(b) Codes
 Circle.m	
 Data.m	
 Fminunc.m	
 Layer.m	
 Nesterov.m	
 Network.m	
 Plotter.m	
 Propagator.m	
 README.txt	
 RMSProp.m	
 SGD.m	
 Trainer.m	 ANNEX_I_Codes_uified_model_language.pdf
 buildModel.m	 Budget.pdf
 datasets.mat	 Report.pdf
(c) Datasets	(d) Documents

Figure 2 Decomposition of the principal folders

2 Code programming and UML

The first step is to define a general structure of the code. When this project started my knowledge about OOP was limited and therefore the initial structure was not the best possible selected. The initial idea was to have three classes: **Data**, **Network** and **Trainer**, being network in composition with Data and Trainer with Network. Sticking to this idea has put up some hindrances during the development of this bachelor's thesis.

The final state of the codes is the shown in figure 3

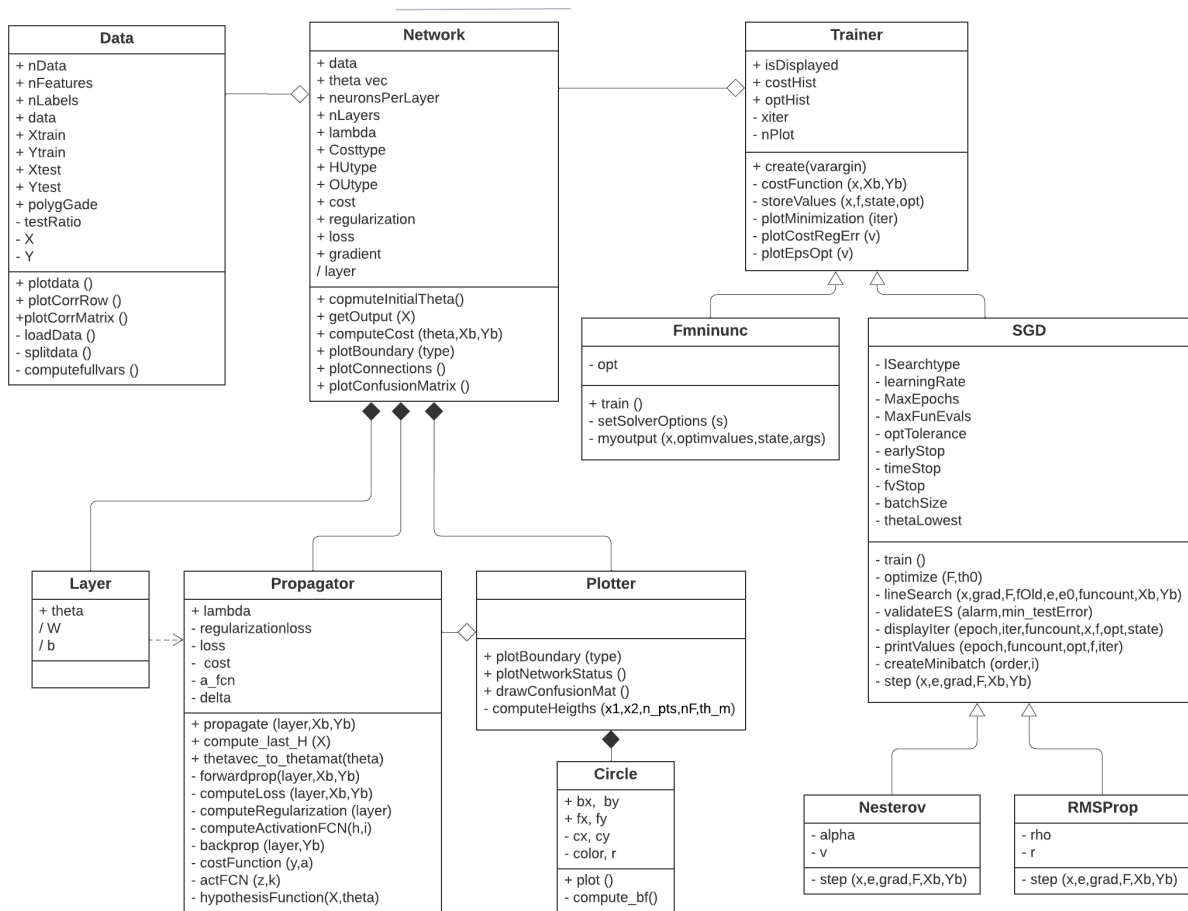


Figure 3 Unified Modeling Language (UML) of the code

3 How to run the code

The idea is to create the 3 main classes (Data, Netowrk and Trainer) and use their methods. Besides the description that will be done in the following paragraphs the best idea to understand the code is to inspect all the files in the folder *Analysis* starting with GeneralCall.m, as all the files in this folder are example scripts using the code.

Initialisation

The initialisation of the 3 main classes have to be summoned as follows:

For class Data

```
data = Data(fn,tr,p)
```

Where *fn* is the name of the file, *tr* is the ratio used for test (30 means 30%), and *p* is the degree of polynomial augmentation to the data desired.

For class Network

It is possible to give 2 or 6 input arguments

```
network = Network(data,structure)
```

```
network = Network(data,structure,costfunction,hiddenunit,outputunit,lambda)
```

Where *data* is an object *Data* with the data wanted to train the ANN with, *structure* is a vector containing the number of neurons at each layer (input and output layer included), *costfunction* is the cost function to use (options: 'loglikelihood' or 'L2'), *hiddenunit* is the activation function desired in the hidden layers (options: 'sigmoid', 'tanh' and 'ReLU'), *outputunit* is the activation function desired in the output layer (options: 'sigmoid' and 'softmax') and *lambda* is a real number (value of the L2 regularization parameter).

With 2 inputs the default parameters are:

```
network = Network(data,structure,' - loglikelihood' , 'ReLU' , 'softmax' , 0);
```

For class Trainer

It is possible to give 4, 7 or 8 input arguments

```
optimizer = Trainer.create(network,algorithm,learningRate,momentum);
```

```
optimizer = Trainer.create(network, algorithm, learningRate, momentum, batch,  
options, linesearch);
```

```
optimizer = Trainer.create(network, algorithm, learningRate, momentum, batch,  
options, linesearch, nplt);
```

Where *network* is the network which has to be trained, *algorithm* is the algorithm used (options: 'SGD', 'Nesterov', 'RMSProp'), *learningRate* is a real number, *momentum* is a real number (alpha for Nesterov and rho for RMSProp), *batch* is an integer (batch size), *options* are a set of minimization criteria, *linesearch* is the type of linesearch (options: 'static', 'decay', 'dynamic', 'fminbnd') and *nplt* is a real number that has to be given if the plot of the minimization is desired which represent every how many epochs to plot.

With 4 inputs the default parameters are:

```
optimizer = Trainer.create(network, algorithm, learningRate, momentum, 200, opt, 'static', 0);
```

```
opt.optTolerance = 1 * 10-6; opt.maxevals = 1000;
```

```
opt.maxepochs = 1000 ; opt.earlyStop = 1000;
```

```
opt.time = Inf([1, 1]); opt.fv = 10-4;
```

Running the methods

Once the initialization has been done the public methods can be called:

`data.plotData()` shows a scatter plot of the data

`data.plotCorrRow(k)` plots the correlation between all the features with feature *k*.

`data.plotCorrMatrix()` plots the whole correlation matrix of the data (if there are lots of features not recommended)

`network.computeInitialTheta()` resets the thetas to a random initial value using a normalized initialization.

`network.getOutput(X)` obtains the prediction given *X* input

`network.plotBoundary(type)` plots the boundary lines if the data has only 2 features, *type* refers to wether 'contour' which gives only the lines or 'filled' which fills the space with color

`network.plotConnections()` plots a diagram of all the neurons and their connections with a range of width and color in their lines based on the importance of the connections (recommended only)

with shallow networks)

`network.plotConfusionMatrix()` plots the confusion matrix

`optimizer.Train()` trains the network with the initialized parameters

4 Conclusions

The code has been successfully implemented, it is extendable and not as easy to read as desirable but sufficient. In the end, when designing big codes is easy to get tangled up. The code can work with 8 different datasets easily and can be extended to work with any dataset if saved in csv. It also does what it was required, and achieves similar levels of accuracy as some libraries of deep learning.